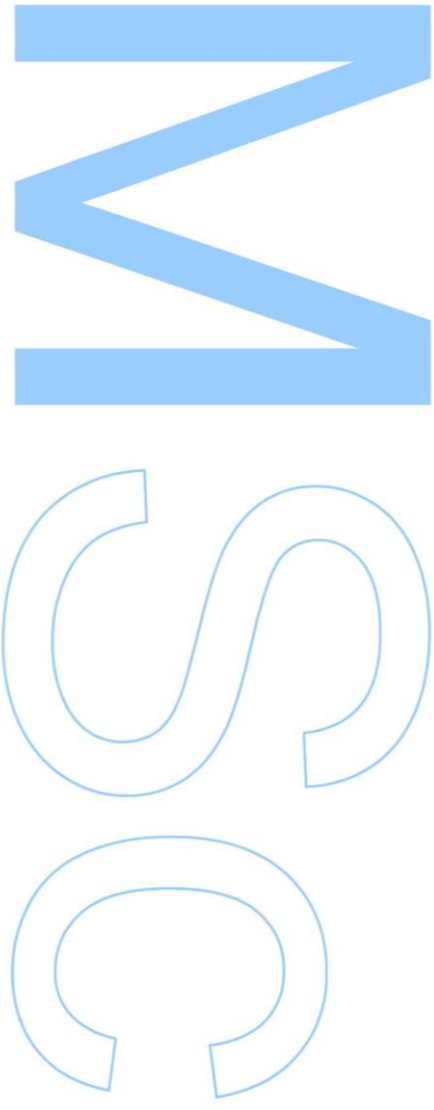
Significant Classes of NFA and their Random Generation

Miguel Ferreira
Mestrado em Ciência de Computadores
Departamento de Ciência de Computadores
2018

Orientador
Rogério Reis, Professor Auxiliar,
Faculdade de Ciências da Universidade do Porto

Coorientador
Nelma Moreira, Professora Auxiliar,
Faculdade de Ciências da Universidade do Porto



U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, _____/_____/_____

To my family.

Acknowledgments

I would like to thank my supervisors, Nelma Moreira and Rogério Reis, for their support and guidance during my degree. I would also like to thank the staff and lecturers of the Department of Computer Science for their consistent will to help. I would like to thank all the friends I've made while I studied in the Faculty of Sciences of the University of Porto, in particular, to the members and coach of the competitive programming team I used to participate on - *Os Ancelistas*: João Ramos, João Pires, Pedro Paredes and Pedro Ribeiro. I also would like to thank my girlfriend, Catarina, for her persistent emotional support and patience. My final and most important acknowledgedment is toward my family for their infinite support.

Abstract

Random generation of non-deterministic finite automata (NFA) can be used to provide experimental average case analysis of algorithms on NFA and as means to randomized unit testing code that uses NFA. We study and implement known methods for random generation of NFA, and for each method we provide experimental results and classify them according to properties a uniform random generator should have.

Important properties of a good random NFA generator include: ability to generate NFA that are non isomorphic, that is, always the same permutation of the NFA is generated; ability to fix the number of states of the generated automata; tractable running time for sampling a large number of instances, e.g. 10000, allowing the average case experiments to meet certain confidence levels; ability to generate automata that are initially connected, i.e. all states are accessible.

Our research resulted in the identification of a specific class of NFA for which the aforementioned important properties are verified. Experimental results showed that this class is almost as succinct as NFA, proving the generator to be a candidate for sampling NFAs for statistically significant experiments.

Resumo

A geração aleatória de autómatos finitos não determinísticos (NFAs) pode ser utilizada para obter análises de caso médio de algoritmos que operam em NFAs e como uma forma de aleatorização de testes unitários que utilizem NFAs. Estudamos e implementamos métodos conhecidos para a geração aleatória de NFAs e, para cada método, obtivemos resultados experimentais e classificações dos métodos de acordo com as propriedades que um gerador aleatório deve ter.

As propriedades de um bom gerador aleatório de NFAs incluem: a habilidade de gerar NFAs que são não isomorfos, isto é, é gerada sempre a mesma permutação do mesmo autómato; a habilidade de fixar o número de estados dos autómatos gerados; um tempo de execução tratável para gerar um número grande de instâncias, p.e. 10000, permitindo que as experiências do caso médio tenham um determinado nível de confiança; a habilidade de gerar autómatos que são inicialmente conexos, isto é, todos os estados são acessíveis.

A nossa investigação resultou na identificação de uma classe específica de NFAs para a qual as propriedades referidas são verificadas. Resultados experimentais sugerem que esta classe é quase tão sucinta como os NFAs, provando assim que o gerador seria um candidato para amostragem de NFAs para experiências statisticamente significantes.

Contents

List of Tables

List of Figures

List of Algorithms

Acronyms

NFA Non-Deterministic Finite Automaton

DFA Deterministic Finite Automaton

FIFA Forward Injective Finite Automaton

ICDFA Initially Connect Complete Deterministic Finite Automaton

TM Turing Machine

CNF Chomsky Normal Form

RPN Reverse Polish Notation

RE Regular Expression

PD Partial Derivative

BFS Breadth First Search

Chapter 1

Introduction

Random generation of NFAs can be used to provide experimental average case analysis of algorithms on NFA and as means to randomized unit testing code that uses NFA. Different methods of NFA generation convey different expected results when assessing average case measures. We study a few methods of random NFA generation, and compare these in terms of performance, bias and uniformity. We also present a novel method of efficiently generating a subclass of NFAs up to isomorphism.

In this chapter we go through the fundamentals for understanding the concepts described in this thesis. We start with concepts of formal languages and automata, some topics in relevant algebraic structures, and some topics in statistics. For more information regarding automata theory we refer the reader to Hopcroft and Ullman's introductory book to automata theory [15].

On Chapter 2 we present some solutions to the random NFA generation problem and expose and compare their advantages and limitations. We also provide new implementations and experimental results of the studied solutions.

On Chapter 3 we present a novel method for generating NFA uniformly up to isomorphism on a specific NFA class. We provide experimental results of typical usages of sampled automata and advocate the advantages and limitations of using such class for random generation. The results of this chapter were published in an international conference proceedings [11].

Finally, on Chapter 4 we conclude that there is empiric evidence that the specific class upon we developed a random generator can be used for significant NFA experimental average case study. This chapter ends with suggestions for future work on open problems regarding the class of automata we defined and some implications of the solution to those problems.

The experimental results presented on this thesis were produced on an Intel® Core™ i7-8550U with 16GB RAM and Ubuntu 18.04 64 bit (Linux 4.4) using Python 2.7 interpreted by Pypy and version 1.3.5.1 of FAdo [10], unless stated otherwise, with 10000 sample instances of the considered objects.

1.1 Set Operations, Relations and Graphs

Let L and R be sets, the relation of inclusion of set L in set R is expressed as $L \subseteq R$ and means that all elements of L are in R, i.e. $(\forall x)(x \in L \Rightarrow x \in R)$. A stronger version of this notion is called proper inclusion, represented by $L \subset R$, meaning that R contains more than just L, i.e. $(\forall x)(x \in L \Rightarrow x \in R \wedge \exists x \in R.x \notin L)$.

The union of two sets L and R, denoted $L \cup R$, is a set containing all elements either in L or R: $L \cup R = \{\ x \mid x \in L \vee x \in R\ \}$.

Dualy, the intersection of two sets L and R is defined as the set containing the elements that simultaneouslly occur in L and R: $L \cap R = \{\ x \mid x \in L \wedge x \in R\ \}$.

Given a domain U we can define the complement of a set S, represented $\overline{S}$, as: $\overline{S} = \{\ x \mid x \in U \wedge x \notin S\ \}$.

The Cartesian product of two sets L and R is the following set of tuples: $L \times R = \{\ (l, r) \mid l \in L \wedge r \in R\ \}$.

The power set of a set S, denoted 2^S, is the set of all subsets of S. Formally: $2^S = \{\ P \mid P \subseteq S\ \}$.

Relation A binary relation R between two sets S and P is a subset of its Cartesian product, i.e. $R \subseteq S \times P$. Two objects x and y are in the relation R if $(x, y) \in R$, denoted xRy.

A binary relation R on a set S can have the following properties:

- Reflexive: $\forall a \in S.aRa$,
- Irreflexive: $\forall a \in S.\neg aRa$,
- Transitive: $\forall a, b, c \in S.aRb \wedge bRc \Rightarrow aRc$,
- Symmetric: $\forall a, b \in S.aRb \Rightarrow bRa$,
- Asymmetric: $\forall a, b \in S.aRb \Rightarrow \neg bRa$.

A relation can be closed under a given property $\mathbb{P}$. The $\mathbb{P}$-closure of a relation R is the smallest relation R' that includes all pairs of R and satisfies $\mathbb{P}$. The transitive closure of R, denoted R^+, is defined by:

- If (a, b) is in R then (a, b) is in R^+,
- If (a, b) is in R^+ and (b, c) is in R^+ then (a, c) is in R^+.

Digraph A digraph, or directed graph, is a tuple (V, E) where V is the set of vertices and the adjacency relation $E \subseteq V \times V$ is the set of edges. Figure 1.1 shows a graph with vertex set $V = \{0, 1, 2\}$ and edge set $E = \{(0,0), (0,1), (0,2), (1,0), (1,1), (2,2)\}$.

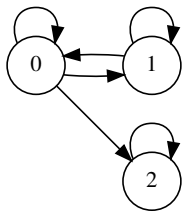


Figure 1.1: Example 3-node graph.

1.2 Finite Automata Theory and Regular Languages

Semigroups and Monoids A semigroup is an algebraic structure consisting of a set and an associative binary operation. A binary operation in a set R is a mapping from $R \times R$ to R. It is associative if, given the binary operation $\circ$:

$$\forall a, b, c \in R.(a \circ b) \circ c = a \circ (b \circ c).$$

A monoid is a semigroup $(R, \circ)$ with an identity element $\varepsilon \in R$, such that:

$$\forall a \in R.\varepsilon \circ a = a \circ \varepsilon = a.$$

Semirings A semiring is a set R with two binary operations $+$ and $\cdot$, called addition and multiplication, such that $(R, +)$ is a commutative monoid with identity element 0, $(R, \cdot)$ is a monoid with identity element 1, the multiplication left and right distributes over addition and 0 is the null element of the multiplication: $a \cdot 0 = 0$.

Strings and Languages An alphabet, typically denoted by Σ, is a set of symbols that compose strings. Strings, or words, are finite sequences of symbols over an alphabet, e.g. on the alphabet $\Sigma = \{a, b\}$, *abba* is a string. The set of all words over an alphabet Σ is denoted $\Sigma^\star$. Any subset L of $\Sigma^\star$ is called a language over the alphabet Σ.

Given two words w and w' one can form the word ww' that results from the juxtaposition of ww', called the concatenation of w and w'. There is a special word $\varepsilon \in \Sigma^\star$ named "empty word" and, for all words w on the alphabet Σ, we have: $\varepsilon w = w\varepsilon = w$.

The length of a string s is denoted $|s|$ and it is the number of symbols that compose the string. It is be defined as $|as| = 1 + |s|,\ a \in \Sigma$ and $|\varepsilon| = 0$.

A language is a set of words over an alphabet Σ. Standard set operations are defined for languages, such as union, intersection and complement, and also the inclusion relation. Moreover, the operation of concatenation and star are defined for languages. Let R and L be two languages, the language $L \cdot R$ is called the concatenation of L and R and is the set $L \cdot R = \{ ww' \mid w \in L \wedge w' \in R \}$, where ww' is the juxtaposition of the words w and w'. Let L be a language, the language $L^\star$ is the called the star, or Kleene closure, of the language and is defined as $L^\star = \cup_{n \geq 0} L^n$, where L^0 is $\{\varepsilon\}$ and $L^n = L^{n-1} \cdot L$. Any language L over Σ is a subset of $\Sigma^\star$, i.e. $L \subseteq \Sigma^\star \iff L$ is a language.

The tuple $(\Sigma^\star, \cdot)$ is a semigroup and, when paired with the empty word ε, forms a monoid.

Regular Expressions Let Σ be an alphabet. The regular expressions over Σ are defined as follows:

- $\emptyset$ is a regular expression denoting the empty language;
- ε is a regular expression denoting the language $\{\varepsilon\}$;
- For each $a \in \Sigma$, a is a regular expression and denotes the language $\{a\}$;
- If r and s are regular expressions representing the languages R and S, respectively, then $(r + s)$ is a regular expression representing the language $R \cup S = \{ w \mid w \in R \vee w \in S \}$;
- If r and s are regular expressions representing the languages R and S, respectively, then (rs) is a regular expression representing the language $RS = \{ ww' \mid w \in R \wedge w' \in S \}$;
- If r is a regular expression representing the language R then $r^\star$ is a regular expression representing the language $R^\star = \bigcup_{i=0}^{\infty} L^i$, where $L^0 = \{\varepsilon\}$ and $L^i = LL^{i-1}$.

The expression $(a + b)^\star a$ represents the language of words over $\{a, b\}^\star$ ending with an a.

A Kleene Algebra is a semiring with idempotent addition and a closure operator $\star$ [17]. The set of regular expressions forms a Kleene Algebra [18].

Deterministic Finite Automaton (DFA) A DFA A is a 5-tuple $\langle Q, \Sigma, \delta, q_0, F \rangle$ where Q is a finite set of states, Σ is a finite set of symbols named alphabet, $\delta : Q \times \Sigma \to Q$ is the transition function, $q_0 \in Q$ and $F \subseteq Q$ are the initial and set of final states, respectively. The transition function δ can be extended to words, i.e. $\delta : Q \times \Sigma^\star \mapsto Q$ with δ inductively defined as, by abuse of notation:

$$\delta(q, aw) = \delta(\delta(q, a), w),$$
$$\delta(q, \varepsilon) = q.$$

The set of words recognized by a DFA $A = \langle Q, \Sigma, \delta, q_0, F \rangle$ constitutes a language $\mathcal{L}(A) = \{ w \mid w \in \Sigma^\star \wedge \delta(q_0, w) \in F \}$. Two DFAs are equivalent if they recognize the same language.

Figure 1.2 contains a DFA recognizing the language of the words ending in a; the final states are represented as double circle nodes; initial states have an inward arrow without source or label; δ is represented as labeled edges of arrows between nodes. The automaton represented is $\langle \{0,1\}, \{a,b\}, \{(0,a,1),(0,b,0),(1,a,1),(1,b,0)\}, 0, \{1\} \rangle$.

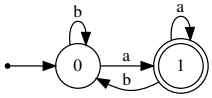


Figure 1.2: DFA recognizing words ending in a.

Any DFA has a unique equivalent with minimal number of states [15]. There is also a known algorithm for minimizing DFA in $O(n \log n)$ time complexity [14]. Comparing the equivalence of the languages recognized by DFA is simply a matter of minimizing both DFAs and comparing if the resulting DFAs are equal up to renaming of the states.

Non-Deterministic Finite Automaton (NFA) An NFA A is a 5-tuple $\langle Q, \Sigma, \delta, I, F \rangle$ where Q is a finite set of states, Σ is a finite set of symbols named alphabet, $\delta : Q \times \Sigma \to 2^Q$ is the transition function, $I \in Q$ and $F \subseteq Q$ are the set of initial and final states, respectively. The transition function δ can be extended to words and sets of states, i.e. $\delta : 2^Q \times \Sigma^\star \mapsto 2^Q$ with δ defined as, by abuse of notation:

$$\begin{aligned} \delta(\emptyset, w) &= \emptyset, \\ \delta(Q, \varepsilon) &= Q, \\ \delta(Q, aw) &= \bigcup_{q \in \delta(Q,a)} \delta(q, w). \end{aligned}$$

The set of words recognized by an NFA $A = \langle Q, \Sigma, \delta, I, F \rangle$ constitutes a language $\mathcal{L}(A) = \{\ w \mid w \in \Sigma^\star \wedge \delta(I, w) \cap F \neq \emptyset\ \}$. Figure 1.3 contains a NFA of the language of the words ending in a; the final states are represented as double circle nodes; initial states have an inward arrow without source or label; δ is represented as labels of arrows between nodes. The automaton represented is $\langle \{0,1\}, \{a,b\}, \{(0,a,0),(0,b,0),(0,a,1)\}, \{0\}, \{1\} \rangle$.

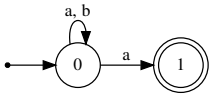


Figure 1.3: NFA recognizing words ending in a.

A DFA is an NFA with $\forall q \in Q,\ \sigma \in \Sigma\ .\ |\delta(q, \sigma)| \leq 1$. An NFA can be converted to a DFA through a determinization algorithm. The complexity of such conversion is known to be $O(2^n)$ [15] and witnesses of worst-case determinization exist for n-state binary NFA [22].

Regular Languages The languages that can be represented by regular expressions are called regular languages. The set of languages denoted by regular expressions, deterministic or non-deterministic finite automata is the same [15].

1.3 Computational Complexity

Big-Oh Notation The efficiency of an algorithm is typically measured as the number of basic operations it performs as a function $T : \mathbb{N} \mapsto \mathbb{N}$ of its input length n. If f, g are two functions from $\mathbb{N}$ to $\mathbb{N}$, the O, Θ, Ω, ω and o notations are defined as follows [3]:

- $(f = O(g)) \Rightarrow (\exists c \in \mathbb{N})(f(n) < c \cdot g(n))$ for sufficiently large n;
- $(f = \Omega(g)) \Rightarrow g = O(f)$;
- $(f = \Theta(g)) \Rightarrow (f = O(g) \wedge g = O(f))$;
- $(f = o(g)) \Rightarrow (\forall \varepsilon > 0)(f(n) \leq \varepsilon \cdot g(n))$ for sufficiently large n;
- $(f = \omega(g)) \Rightarrow g = o(f)$.

To emphasize the input parameter, it is often written $f(n) = O(g(n))$ instead of $f = O(g)$ and similarly for Θ, Ω, ω and o.

Turing Machine Model A Turing machine is a simple mathematical model for the formalization of an effective procedure [15]. The Churchs's hypothesis suggests that this model is equivalent to our intuitive notion of a computer.

Formally, a (deterministic) Turing machine (TM) is denoted

$$M = \langle Q, \Sigma, \mathrm{T}, \delta, q_0, B, F \rangle$$

where

- Q is the finite set of states;
- Σ, a subset of T not including B, is the set of input symbols;
- T is the finite set of allowable table symbols;
- δ is the transition function, a mapping from $Q \times \mathrm{T}$ to $Q \times \mathrm{T} \times \{L, R\}$;
- B, a symbols of T is the blank symbol;
- $F \subseteq Q$ is the set of accepting states.
- q_0 in Q is the initial state;

On every move a TM, depending on the symbol scanned on the head and the current state of the finite control:

- changes state,
- prints a symbol on the tape cell scanned, replacing what was written there, and
- moves its head left or right one cell, on the tape that is unlimited to the right.

The instantaneous description (ID) of a TM on the current state q, where α_1, α_2 are strings in $\mathrm{T}^\star$ that are the contents of the tape up to the head and from the head to the rightmost non-blank symbol, is $\alpha_1 q \alpha_2$.

A TM move is defined as follows. Let $X_1 X_2 \cdots X_{i-1} q X_i \cdots X_n$ be the ID of the TM. We have two possible directions: if $\delta(q, X_i) = (p, Y, L)$ then we write $X_1 X_2 \cdots X_{i-1} q X_i \cdots X_n \vdash_M X_1 X_2 \cdots Y p X_{i-1} X_{i+1} \cdots X_n$; if $\delta(q, X_i) = (p, Y, R)$ then we write $X_1 X_2 \cdots X_{i-1} q X_i X_{i+1} \cdots X_n \vdash_M X_1 X_2 \cdots X_{i-1} Y p X_{i+1} \cdots X_n$. If one ID a results from a finite number of moves from b we write $b \vdash_* a$. The execution time for a halting run of a word is the number of moves of the TM.

The language accepted by a deterministic Turing machine $M = \langle Q, \Sigma, \mathrm{T}, \delta, q_0, B, F \rangle$ is denoted $L(M)$ and is:

$$L(M) = \{\ w \mid w \in \Sigma^\star \wedge q_0 w \vdash_* \alpha_1 p \alpha_2, p \in F, \alpha_1 \in \mathrm{T}^\star, \alpha_2 \in \mathrm{T}^\star\ \}$$

A TM can be non-deterministic if we allow δ to perform several moves from the same source ID in a single computation step. This computation model recognizes the same languages as the deterministic TM.

The language accepted by a non-deterministic Turing machine $M = \langle Q, \Sigma, \mathrm{T}, \delta', q_0, B, F \rangle$ where $\delta' : Q \times \mathrm{T} \mapsto 2^{Q \times \mathrm{T} \times \{L,R\}}$ are the words for which there is a witness - a finite sequence of choices of one of the moves for each non-deterministic step - the word that leads the non-deterministic TM into an accepting state.

Basic Complexity Classes A complexity class is a set of functions that can be computed within given resource bounds. Let $T : \mathbb{N} \mapsto \mathbb{N}$ be a total function. A language L is in DTIME($T(n)$) if and only if there is a Turing machine that runs in time $O(T(n))$ and decides L. Deciding a language $L \subseteq \{0,1\}^\star$ is equivalent to computing a function $f_L : \{0,1\}^\star \mapsto \{0,1\}$ where $f_L(x) = 1 \Leftrightarrow x \in L$ [3].

The class P is the class of problems decided by deterministic Turing machines with a running time bounded by a polynomial:

$$\mathrm{P} = \bigcup_{c \geq 1} \mathrm{DTIME}(n^c)$$

Let $T : \mathbb{N} \mapsto \mathbb{N}$ be a total function. A language L is in NTIME($T(n)$) if and only if there is a non-deterministic Turing machine that runs in time $c \cdot O(T(n))$ for some constant $c > 0$

and decides L. The class of NP is the class of languages decidable in polynomial time by a non-deterministic Turing machine [3]:

$$\mathrm{NP} = \bigcup_{c \geq 1} \mathrm{NTIME}(n^c)$$

Whether P = NP is an open problem and of great importance in computer science and mathematics.

1.4 The Graph Isomorphism Problem

Digraph Isomorphism Two directed graphs $G = (V, E)$ and $H = (V', E')$ are isomorphic if there is a bijection $f : V \mapsto V'$ such that $(s, t) \in E$ if and only if $(f(s), f(t)) \in E'$. When G and H are the same digraph, f is called an automorphism of G. The generic graph isomorphism problem is applied to undirected graphs but we will focus on the isomorphism of graphs and NFAs.

Two NFAs are isomorphic if there is a bijection φ between their sets of states preserving the set of initial states, final states and transitions. Let $A = \langle Q, \Sigma, \delta, I, F \rangle$ be an NFA, $\varphi(A) = \langle \varphi(Q), \Sigma, \{\ (\varphi(s), k, \varphi(t)) \mid (s, k, t) \in \delta\ \}, \varphi(I), \varphi(F) \rangle$.

Figure 1.4: Two 3-node isomorphic graphs.

It is not known if the graph isomorphism problem is solvable in polynomial time but it is known that it is not NP-complete unless P = NP [24]. Isomorphism of specific classes of graphs and automata are known to be polynomially solvable such as fixed bound degree graphs [20] and initially connected DFAs [1]. However, it is known that the general graph isomorphism problem for graphs, semigroups and automata (NFAs) are polynomially equivalent [6]. Moreover, the graph isomorphism problem for digraphs polynomially reduces to general graph isomorphism [23].

This means that testing isomorphism of generic NFA, for the purpose of random generation, can be computationally expensive, although recent studies suggest that the generic problem for n-node graphs can be solved, in theory, in quasipolinomial time - $2^{O((\log n)^3)}$ [4, 13].

Let $\mathbb{C}_n$ be a class of n-node digraphs closed by isomorphism. There are $n!$ possible bijections on $\{1,\ldots,n\}$. The automorphism group of a graph G is denoted $\mathrm{Aut}(G)$. The number of isomorphism classes of a graph G is $\frac{n!}{|\mathrm{Aut}(G)|}$. We will present some random generators of automata that are uniform with respect of the isomorphism classes for n-state k-symbol finite automata.

On the finite automata case, a practical solution for calculating the size of the automorphism group of an NFA consists in using state labels [12].

Algorithm 1 NFA automorphism group size using labelings

```
n
1: procedure CountAutomorphism(NFA A, labeling τ with image D = {α_1, ..., α_l})
2:     size ← 0
3:     for α ∈ D do
4:         C[α] ← ∅                                   ▷ precomputed τ^{-1}
5:     for i ∈ {1, ..., n} do
6:         C[τ(A, i)] = C[τ(A, i)] ∪ {i}
7:     for each permutation ρ_1 of C(α_1) do
8:         for each permutation ρ_2 of C(α_2) do
9:             ...
10:            for each permutation ρ_l of C(α_l) do
11:                if ρ = ρ_1 ... ρ_l ∈ Aut(A) then
12:                    size ← size + 1
13:    return size
```

Let $\mathbb{A}_n$ be a class of n-state NFA, a labeling is a computable function $\tau : \mathbb{A}_n \times \{1,\ldots,n\} \mapsto D$, such that if φ is an isomorphism between two NFA A and B, then for every $i \in \{1,\ldots,n\}, \tau(A,i) = \tau(B,\varphi(i))$. Algorithm 1 consists in finding functions φ that preserve the labeling τ. Commonly used labels for NFA include states being final or not and states being initial or not.

1.5 Statistics

Sample Size Determination Because the size of the population of n-state k-symbol automata grows very quickly it is necessary to take a statistically significant sample in order to estimate average properties of the population.

By accepting an interval of error in our measure and a probability of the true value of the property being within the confidence interval, i.e. the range of values of the measure in our sample considering the accepted error, it is possible to estimate a sample size that satisfies the accepted frequencies of error. The interval of error accepted in our measure is called margin of error ε and the probability of the true value being within the confidence interval is called confidence level γ.

According to [9], given a margin of error ε and an estimated proportion p of the population

we are interested in, we can obtain the sample size $N = \frac{z^2 p(1-p)}{\varepsilon^2}$ where z is abscissa of the normal curve such that for a confidence level γ: $P(-z < Z < z) = \gamma$, where Z is our standardized measure. According to the central limit theorem, assuming the data is uniformly distributed and because we know the population is large, we can assume $p = \frac{1}{2}$ for maximum variability as we do not know the proportion of classes that are interesting to the statistic beforehand. Therefore, we obtain the equation $N = \frac{z^2}{4\varepsilon^2}$. Refer to Table 1.1 for common values of confidence levels and respective sample sizes used in sampling, accepting a 1% error margin.

γ	N
0.99	16590
0.95	9604
0.90	6765

Table 1.1: Samples sizes in funcion of confidence level with a 1% error margin.

Markov Chains A Markov chain in a finite set Ω is a sequence $X_0, \ldots, X_t$ of random variables on Ω such that $\mathbb{P}(X_{t+1} = x_{t+1} \mid X_t = x_t) = \mathbb{P}(X_{t+1} = x_{t+1} \mid X_t = x_t, \ldots, X_i = x_i, \ldots, X_0 = x_0), \forall x_i \in \Omega$. The probability of the next element of the sequence depends only on the previous element and not on the whole sequence.

A Markov chain is defined by a transition matrix M, which is a transition function from $\Omega \times \Omega$ into $[0, 1]$ such that $M(x, y) = \mathbb{P}(X_{t+1} = y \mid X_t = x)$. The transition matrix describes a directed graph of the Markov chain and there is an edge from x to y if $M(x, y) \neq 0$ and $\forall x \in \Omega \ \sum_{y \in \Omega} M(x, y) = 1$.

$$M = \begin{bmatrix} 0.9 & 0.1 \\ 0.8 & 0.2 \end{bmatrix}$$

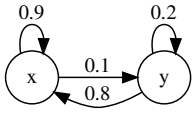


Figure 1.5: Example of a Markov chain digraph and corresponding transition matrix.

A Markov chain is irreducible if its digraph is strongly connected. It is aperiodic if for all nodes $x \in \Omega$, the greatest common divisor of the length of all the cycles visiting x is 1. In particular, if $M(x, x) \neq 0$ then the Markov chain is aperiodic.

A Markov chain M is ergodic if it is irreducible and aperiodic and it is symmetric if $\forall x, y \in \Omega . M(x, y) = M(y, x)$. It is known that an ergodic chain has a unique stationary distribution π, i.e. $\pi M = \pi$ and if the chain is symmetric this distribution is uniform in Ω.

Let π be a stationary distribution on M, sampling random elements of Ω according to the distribution π is known as the Monte-Carlo method. This method consists in choosing arbitrarily X_0 and computing a sequence $X_1, X_2, \ldots, X_t$ for t large enough. Calculating t can be done

using the concept of ε-mixing time: $t_{mix}(\varepsilon) = min\{\ t \mid max_{x\in\Omega}||M_x^t - \pi||_{TV} \leq \varepsilon\ \}$, where $||P - Q||_{TV} = \sup_{A\in\Omega} |P(A) - Q(A)|$ is the total variation distance of the probability measure.

The Metropolis-Hastings algorithm is based on the Monte-Carlo technique and consists in generating random elements of Ω by modifying the transition function in order to obtain a particular stationary distribution ν. The Metropolis-Hastings chain P_ν is defined as:

$$P_\nu(x,y) = \begin{cases} \min\{1, \frac{\nu(y)}{\nu(x)}\}M(x,y) & \text{if } x \neq y \\ 1 - \sum_{z\neq x} \min\{1, \frac{\nu(z)}{\nu(x)}\}M(x,z) & \text{if } x = y \end{cases}.$$

Chapter 2

Random Generation of NFA

The average-case analysis of operations in automata is, in general, a difficult task. One approach to this problem is to consider uniformly distributed random representations and to perform statistically significant experiments requiring most of the times nonisomorphic sampled automata. Specific classes of automata for which this problem was solved have been studied, e.g. initially connected complete DFAs [1, 5]. However for NFAs, the problem seems unfeasible in general as for n-state NFAs the size of the automorphism group can be $n!$, and this is polynomially equivalent to testing if two NFAs are isomorphic.

The importance of the uniform distribution with respect to isomorphism classes when sampling NFA is important because some automata can have less isomorphic representatives then other automata, and because generators typically generate a specific permutation of a certain automaton, e.g. the initial state always gets name *0*, we need to generate this instance with a probability weighted by the size of isomorphism class of the automaton.

In this chapter we will describe three distinct models of random NFA generation, including one that is uniform up to isomorphism, which we implemented and provide comparative results.

2.1 Random Generation of Regular Expressions

A context free grammar (CFG) G is a 4-tuple $G = \langle V, \Sigma, R, S\rangle$ where V is the set of non-terminal symbols, Σ is the set of terminal symbols, $R \subseteq V \times (V \cup \Sigma)^\star$ is the set of production rules and $S \in V$ is the starting symbol. A production rule A is written as $A \to \alpha$, $A \in V$, $\alpha \in (V \cup \Sigma)^\star$. A word $w \in \Sigma^\star$ belongs to the language recognized by a CFG if there is a derivation - a sequence of rule applications - starting in $S \in V$ and by applying production rules to non-terminals until there is no terminals left w is yielded. The existence of such derivation is denoted $S \Rightarrow w$. See [15] for more details.

Given a context-free grammar G for regular expressions and an integer n, one can generate random words e of size n which are regular expressions [21]. Each random regular expression e can

be converted to an NFA. The ratio between sizes of regular expressions and their corresponding constructed NFA has been studied [7].

We assume a grammar G in Chomksy normal form (CNF) without ε-productions except for the start symbol, so that all productions are of the form $A \to BC$ or $A \to a$, where A, B, C are non-terminal and $a \in \Sigma$ is a terminal. It is possible to convert any context-free grammar to Chomksy normal form [15]. For each non-terminal A, define $||A||_l = |\{\ w \mid A \Rightarrow w \text{ and } |w| = l\ \}|$ and $||A \to BC||_l = |\{\ w \mid BC \Rightarrow w \text{ and } |w| = l\ \}|$. Algorithm 2 implements the simple idea behind sampling words from a CNF grammar uniformly at random, by counting the number of l-sized words for each production of the grammar. The word generation phase, for length l and nonterminal A, consists in choosing a production $A \to BC$ with probability $\frac{||A \to BC||_l}{||A||_l}$, and a split $0 < k < l$ chosen with probability $\frac{||B||_k||C||_{l-k}}{||A \to BC||_l}$. This split generates uniformly at random two words b and c with lengths k and $l-k$, respectively, that concatenated form our uniformly random l-sized word.

Algorithm 2 Number of l-sized words for each production of a CNF grammar

```
1: procedure CountWords(Productions in CNF)
2:     for each nonterminal A do
3:         Count[A, 1] ← 0
4:     for each production A → a do
5:         Count[A, 1] ← Count[A, 1] + 1
6:     for l ← 2, up to n do
7:         for each nonterminal A do
8:             for each production A → BC do
9:                 Count[A, l] ← Count[A, l] + Σ_{0<k<l} Count[B, k].Count[C, l − k]
       return Count
```

Table 2.1 shows some properties of the partial derivative automaton [2] constructed from a set of 10000 random regular expressions with of n length in symbols and operators and k possible alphabet symbols *usefulS*: number of states of the partial derivative automaton; T: number of transitions of the partial derivative automaton; *dfa*: percentage of NFA that are DFA; *minS*: number of states of the minimal DFA; *minT*: number of transitions of the minimal DFA; *subsetS*: number of states of the subset constructed DFA; *subsetT*: number of transitions of the subset constructed DFA; *timeg*: seconds required to randomly generate the all the 10000 instances; *timep*: second required to process the 10000 instances. The context free grammar used was the RPN for regular expressions, to avoid parenthesis being counted as symbols:

$$e \to +\ e\ e \mid .\ e\ e \mid *\ e \mid a \mid \varepsilon \mid \emptyset \qquad\qquad a \in \Sigma.$$

For the purposes of random generation of NFA, this generator is typically paired with a NFA construction method from regular expressions. Although generating fixed size regular expressions uniformly according to their valid infix or polish notations, from a given context-free grammar, seems like a good bias, the choice of the construction method will generate NFA with predictable properties depending on the construction method.

(n, k)	dfa	minS	subsetS	minT	subsetT	usefulS	T	timeg	timep
(10, 2)	0.48	2.45	3.14	2.76	3.71	3.15	4.23	0.51	4.56
(10, 3)	0.54	2.91	3.45	3.61	4.48	3.39	4.81	0.22	2.77
(10, 4)	0.60	3.16	3.60	4.12	4.89	3.52	5.09	0.20	2.62
(10, 10)	0.78	3.71	3.92	5.29	5.72	3.84	5.73	0.18	2.45
(20, 2)	0.07	3.42	4.93	4.75	7.44	5.06	10.31	0.57	7.99
(20, 3)	0.10	4.49	5.74	7.16	9.97	5.47	11.35	0.51	6.94
(20, 4)	0.15	5.03	6.09	8.54	11.22	5.71	11.87	0.30	6.10
(20, 10)	0.42	6.10	6.57	11.22	12.71	6.24	12.42	0.46	5.14
(30, 2)	0.01	4.12	6.65	6.34	11.12	6.91	17.81	0.79	14.66
(30, 3)	0.02	5.92	8.20	10.80	16.56	7.51	19.43	0.55	14.55
(30, 4)	0.03	7.06	8.88	13.92	19.25	7.90	19.97	0.59	14.06
(30, 10)	0.20	8.58	9.37	18.25	21.27	8.64	19.94	0.75	10.99
(50, 2)	0.00	5.10	10.13	8.64	18.52	10.67	36.58	1.11	51.64
(50, 3)	0.00	9.12	13.89	19.21	32.33	11.60	38.59	1.22	51.48
(50, 4)	0.00	11.45	15.34	26.81	39.77	12.21	38.69	1.33	41.83
(50, 10)	0.04	13.82	15.36	35.23	42.59	13.37	36.93	1.16	37.94
(100, 2)	0.00	6.23	19.67	11.30	38.10	20.00	97.91	6.28	767.16
(100, 3)	0.00	18.03	34.26	44.33	91.73	21.88	98.42	6.56	700.79
(100, 4)	0.00	25.29	38.43	72.49	121.15	22.98	96.08	6.61	650.24
(100, 10)	0.00	28.93	32.77	92.83	115.41	25.25	83.55	6.48	448.07

Table 2.1: Average measures of partial derivative NFA generated from 10000 random regular expressions of RPN length n with k symbols.

The random regular expression method has the advantage of being versatile and easily adaptable for non-regular languages because it is based on a random word generator for context-free languages.

2.2 Generation by Bitstreams

This method was used by van Zijl to generate random NFA [8, 25]. Bitstream NFA generation consists in, according to the algorithm, given an alphabet $\Sigma = \{0, \ldots, k-1\} = [0, k[$ and the set of states $Q = \{0, \ldots, n-1\} = [0, n[$:

- generating a uniform random string of bits of size kn^2. If the bit at position $ln^2 + in + j + 1$ is non-zero then there is a transition from state i to state j with label l.
- the state 1 is the single initial state.
- the set of final states is equiprobably chosen from the subsets of Q.

Algorithm 3 implements the bitstream NFA generator assuming that RandomBits(n) returns a list of n random binary digits. Figure 2.1 shows a correspondence between a bitstream and a 2-state 2-symbol NFA.

Algorithm 3 Sampling of n-state k-symbol NFA using bitstreams

```
1:  procedure BitstreamRandomNFA(n = |Q|, k = |Σ|)
2:      δ ← ∅
3:      b ← RandomBits(kn²)
4:      for l ← 0 to k − 1 do
5:          for i ← 0 to n − 1 do
6:              for j ← 0 to n − 1 do
7:                  if b[ln² + in + j + 1] = 1 then
8:                      δ ← δ ∪ {(i, l, j)}
9:      F ← ∅
10:     f ← RandomBits(n)
11:     for i ← 0 to n − 1 do
12:         if f[i] = 1 then
13:             F ← F ∪ {i}
        return NFA ([0, n[, [0, k[, δ, {0}, F)
```

This generation model has the advantage of being simple and efficient, since it has $O(kn^2)$ time and space complexity, but comes with many disadvantages. The objects generated by this method have the disadvantage of not being necessarily initially connected neither uniform with respect to the set of automorphism groups of the n-state k-symbol NFAs. We provide some properties of the NFAs from our implementation of the generator in Table 2.2. Our experiment consisted in sampling 10000 NFAs using the van Zijl method and calculating: *dfa* - proportion of the generated NFAs that are a DFA; *minS* - average number of states of the minimal DFA; *subsetS* - average number of states of the subset constructed DFA; *usefulS* - average number of reachable useful states; T - average number of transitions; *timeg* - time required to sample the 10000 NFAs; *timep* - time required to calculate properties on the 10000 NFAs.

l	1				2			
bit	1	1	0	0	0	0	1	1
i,j	1,1	1,2	2,1	2,2	1,1	1,2	2,1	2,2
idx	1	2	3	4	5	6	7	8

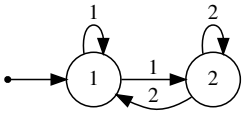


Figure 2.1: Example of a bitstream to 2-state automaton correspondence.

A derived approach from the van Zijl one was proposed in [19], with the purpose of studying the size of DFAs obtained by determinization in function of their transition density. It generates NFAs by first randomly generating a connected structure, in order to end up with an accessible automaton; a unique single state is chosen and then the transitions are randomly chosen: if a transition appears twice, it is rejected and another one is chosen. The automata are generated for a fixed number of states n, number of alphabet symbols k and number of transitions e. The

non-deterministic density d is given by $d = \frac{e}{mn^2}$.

This generation method has the advantage of having the same time and space complexity of the bitstream generator $O(kn^2)$, and also the guarantee that the generated objects are initially connected. Algorithm 4 implements density based NFA generator by assuming that $\textsc{Random}(S)$ returns a random element from set S and $\textsc{Random}(a, b)$ returns a uniform random real number in $[a, b]$.

We provide some properties of the NFAs from our implementation of the generator in Table 2.3. Our experiment consisted in sampling 10000 NFAs using the method for initially connected NFAs using bitstreams with 50% transition density and calculating: *dfa* - proportion of the generated NFAs that are a DFA; *minS* - average number of states of the minimal DFA; *subsetS* - average number of states of the subset constructed DFA; *usefulS* - average number of reachable useful states; T - average number of transitions; *timeg* - time required to sample the 10000 NFAs; *timep* - time required to calculate properties on the 10000 NFAs.

Algorithm 4 Sampling of n-state k-symbol NFA using density

```
1:  procedure LeslieRandomNFA(n = |Q|, k = |Σ|, t transitions)
2:      δ ← InitiallyConnect([0, n[, [0, k[)
3:      d ← (t−n+1)/(n²k)
4:      for s ∈ [0, n[ do
5:          for σ ∈ [0, k[ do
6:              for t ∈ [0, n[ do
7:                  r ← Random(0, 1)
8:                  if r ≤ d then
9:                      δ ← δ(s, σ, t)
10:     F ← ∅
11:     f ← RandomBits(n)
12:     for i ← 0 to n − 1 do
13:         if f[i] = 1 then
14:             F ← F ∪ {i}
        return NFA ([0, n[, [0, k[, δ, {0}, F)
15: procedure InitiallyConnect(Q state set, Σ symbol set)
16:     V ← {0}
17:     δ ← ∅
18:     while V ≠ Q do
19:         s ← Random(V)
20:         t ← Random(Q \ V)
21:         k ← Random(Σ)
22:         δ ← δ ∪ (s, k, t)
23:         V ← V ∪ {t}
        return δ
```

(n, k)	dfa	minS	subsetS	minT	subsetT	usefulS	T	timeg	timep
(5, 2)	0.03	1.43	3.77	2.72	7.42	4.70	23.40	0.28	1.48
(5, 3)	0.03	1.69	5.13	4.85	15.22	4.74	35.44	0.18	1.39
(5, 10)	0.03	7.35	14.44	72.47	143.51	4.74	118.06	0.47	12.55
(10, 2)	0.00	1.06	2.71	2.11	5.42	9.98	99.90	0.31	1.31
(10, 3)	0.00	1.06	3.10	3.17	9.30	9.99	149.72	0.46	2.03
(10, 10)	0.00	1.07	5.77	10.69	57.66	9.99	499.21	1.25	12.20
(15, 2)	0.00	1.01	2.29	2.03	4.58	15.00	225.17	0.75	2.55
(15, 3)	0.00	1.02	2.44	3.04	7.32	15.00	337.38	1.22	3.72
(15, 10)	0.00	1.01	3.46	10.13	34.55	15.00	1124.69	2.94	15.21
(25, 2)	0.00	1.00	2.04	2.00	4.07	25.00	624.76	1.60	5.82
(25, 3)	0.00	1.00	2.05	3.00	6.16	25.00	937.29	2.25	7.95
(25, 10)	0.00	1.00	2.17	10.01	21.71	25.00	3124.89	9.18	30.88

Table 2.2: NFA properties with bitstream generation.

(n, k)	dfa	minS	subsetS	minT	subsetT	usefulS	T	timeg	timep
(5, 2)	0.03	1.57	4.20	2.98	8.26	4.70	21.79	0.65	3.15
(5, 3)	0.03	1.90	5.69	5.47	16.86	4.73	33.59	0.31	2.89
(5, 10)	0.03	8.34	15.21	82.20	151.01	4.73	115.73	0.87	27.49
(10, 2)	0.00	1.06	2.82	2.11	5.63	9.97	95.49	0.73	2.76
(10, 3)	0.00	1.06	3.20	3.17	9.60	9.98	145.56	0.97	4.19
(10, 10)	0.00	1.08	6.01	10.75	60.14	9.98	494.51	2.30	24.95
(15, 2)	0.00	1.01	2.33	2.03	4.66	15.00	218.62	1.17	4.65
(15, 3)	0.00	1.01	2.48	3.04	7.45	15.00	330.86	1.72	7.43
(15, 10)	0.00	1.01	3.48	10.13	34.83	15.00	1118.16	5.38	31.21
(25, 2)	0.00	1.00	2.04	2.00	4.08	25.00	613.39	3.09	10.58
(25, 3)	0.00	1.00	2.06	3.00	6.18	25.00	925.81	4.37	15.97
(25, 10)	0.00	1.00	2.17	10.01	21.71	25.00	3112.51	14.92	54.26

Table 2.3: NFA properties with bitstream and density generation.

2.3 Generation by Markov Chains

Several families of NFA which have symmetric ergodic Markov chains are proposed by Héam and Pierre-Cyrille [12]. They define a Metropolis-Hastings Markov chain for a class of automata $\mathbb{C}$ by the transition matrix $S^{\mathbb{C}}_{\rho_1,\rho_2,\rho_3}(x,y)$, where ρ_1, ρ_2, ρ_3 are three real numbers satisfying $0 \leq \rho_i \leq 1$ and $\rho_1 + \rho_2 + \rho_3 = 1$:

- $S^{\mathbb{C}}_{\rho_1,\rho_2,\rho_3}(x,y) = \frac{\rho_1}{|Q|}$ if y is the automaton x with setting/unsetting a state initial,

- $S^{\mathbb{C}}_{\rho_1,\rho_2,\rho_3}(x,y) = \frac{\rho_2}{|Q|}$ if y is the automaton x with setting/unsetting a state final,
- $S^{\mathbb{C}}_{\rho_1,\rho_2,\rho_3}(x,y) = \frac{\rho_3}{|\Sigma||Q|^2}$ if y is the automaton x by adding/removing a single transition $(p,a,q) \in Q \times \Sigma \times Q$,
- $S^{\mathbb{C}}_{\rho_1,\rho_2,\rho_3}(x,x) = 1 - \sum_{y \neq x} S^{\mathbb{C}}_{\rho_1,\rho_2,\rho_3}(x,y)$,
- otherwise $S^{\mathbb{C}}_{\rho_1,\rho_2,\rho_3}(x,y) = 0$.

Let $\mathbb{C}_n$ be a class of n-state k-symbol automata closed by isomorphism and γ_n the number of isomorphism classes on $\mathbb{C}_n$:

Proposition 1 ([12]). *Randomly generating an element x of $\mathbb{C}_n$ with probability $\frac{|Aut(x)|}{\gamma_n n!}$ provides a uniform random generator of the isomorphism classes of $\mathbb{C}_n$.*

Proof *Let H be an isomorphism class of $\mathbb{C}_n$, H is generated with probability:*

$$\sum_{x \in H} \frac{|Aut(x)|}{\gamma_n n!} = \sum_{x \in H} \frac{1}{\gamma_n |H|} = \frac{1}{\gamma_n |H|} \sum_{x \in H} 1 = \frac{|H|}{\gamma_n |H|} = \frac{1}{\gamma_n}.$$

□

It is not necessary to calculate γ_n in order to compute the Metropolis-Hastings Markov chain, it is just necessary to calculate $\frac{\nu(x)}{\nu(y)} = \frac{|\mathrm{Aut}(y)|}{|\mathrm{Aut}(x)|}$. We obtain the following Markov chain that uniformly generates automata up to isomorphism on a class $\mathbb{C}$ where ρ_1 is the probability of changing an initial state, ρ_2 is the probability of changing a final state and ρ_3 is the probability of adding or removing a transition:

$$M^{\mathbb{C}}_{\rho_1,\rho_2,\rho_3}(x,y) = \begin{cases} \min\{\ 1, \frac{|\mathrm{Aut}(y)|}{|\mathrm{Aut}(x)|}\ \} S^{\mathbb{C}}_{\rho_1,\rho_2,\rho_3}(x,y) & \text{if } x \neq y \\ 1 - \sum_{z \neq x} \min\{\ 1, \frac{|\mathrm{Aut}(z)|}{|\mathrm{Aut}(x)|}\ \} S^{\mathbb{C}}_{\rho_1,\rho_2,\rho_3}(x,z) & \text{if } x = y \end{cases}.$$

We provide some properties of the NFA produced by implementation of this generator in Table 2.4. Our experiment consisted in sampling 10000 NFAs using the Markov chain method, where: *dfa* - proportion of the generated NFAs that are a DFA; *minS* - average number of states of the minimal DFA; *subsetS* - average number of states of the subset constructed DFA; *usefulS* - average number of reachable useful states; *T* - average number of transitions; *timeg* - seconds required to sample the 10000 NFAs; *timep* - seconds required to calculate properties on 10000 NFAs. We used the values: $\rho_1 = \frac{n}{2n+kn^2}$, $\rho_2 = \frac{n}{2n+kn^2}$, $\rho_3 = 1 - \rho_1 - \rho_3$ and the class $\mathbb{U}$ of the universe of all n-state k-symbol NFA.

This generation method has the advantage of being versatile, applicable to several automata classes and generating objects up to isomorphism. However, it requires the user to provide the probabilities for changes in the automaton, requires the calculation of a mixing time since it is a Monte-Carlo sampling method and each movement in the modified Markov chain requires $O(kn^2)$

(n, k)	dfa	minS	subsetS	minT	subsetT	usefulS	T	timeg	timep
(2, 2)	0.46	1.41	1.57	1.26	1.48	1.64	1.39	27.30	1.37
(2, 3)	0.40	1.65	1.81	2.23	2.53	1.71	2.10	29.34	0.40
(2, 5)	0.32	2.00	2.15	4.04	4.40	1.79	3.16	48.37	0.48
(3, 2)	0.18	1.98	2.67	2.83	4.07	2.47	4.31	184.54	0.67
(3, 3)	0.12	3.01	3.70	6.55	8.28	2.67	6.56	253.91	0.74
(3, 5)	0.06	4.57	4.95	15.53	16.97	2.84	9.90	423.16	1.01
(5, 2)	0.01	1.42	4.13	2.62	7.87	4.86	19.60	2786.02	0.70
(5, 3)	0.00	2.57	6.92	7.19	19.71	4.96	28.26	8988.35	1.35
(5, 5)	0.00	11.74	16.82	54.81	78.89	5.00	41.65	19364.78	7.37

Table 2.4: NFA properties with Markov chain generation.

computations of automorphism group sizes of NFA. Furthermore, it is proved that for output bound degree NFAs, be it by maximum number of outgoing transitions or maximum number of reached states, the isomorphism test problem of NFAs has polynomial time complexity [12], regardless of having a single or multiple initial states.

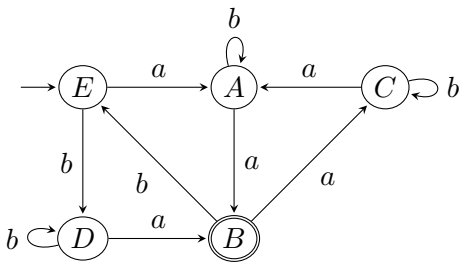
2.4 Initially Connected Complete DFA

A DFA $A = \langle Q, \Sigma, \delta, q_0, F\rangle$ is initially connected if for every state $q \in Q$ there is a word $w \in \Sigma^\star$ such that $\delta(q_0, w) = q$; it is complete if $\delta(q, \sigma)$ is defined for every state $q \in Q$ and symbol $\sigma \in \Sigma$.

We suppose that $||\Sigma|| = k$ and that its elements are ordered some natural order. Given an ICDFA, initially connected complete DFA, $A = \langle Q, \Sigma, \delta, q_0, F\rangle$, a canonical order for Q can be obtained by a breath-first traversal of A, starting with the initial state q_0, and in each state considering the transitions in the natural order of Σ.

There are several studies of the uniform random generation for this class of DFA [1, 5]. Let ICDFA$_\emptyset$ $A_\emptyset = \langle Q, \Sigma, \delta, q_0\rangle$ be the semiautomaton of an ICDFA $A = \langle Q, \Sigma, \delta, q_0\rangle$, i.e. the automaton discarding final states.

Example 2.4.1. Consider the ICDFA, $A = \langle \{A, B, C, D, E\}, \{a, b\}, \delta, E, \{B\}\rangle$, $a < b$, and φ the states renaming according to the induced order.



φ	A	B	C	D	E
	1	3	4	2	0

.

The canonical string $S(A)$ for the corresponding ICDFA$_\emptyset$ is $\underbrace{12}_{E}\ \underbrace{31}_{A}\ \underbrace{32}_{D}\ \underbrace{40}_{B}\ \underbrace{14}_{C}$. Adding the information pertaining to the final states as a sequence of bits, the canonical string for A is 123132401400010.

Uniform random generation of ICDFA up to isomorphism is done by first generating a random uniform ICDFA$_\emptyset$ up to isomorphism and a random uniform subset of Q for the set of final states. The following lemmas are hold [1]:

Lemma 2.4.1 ([1])**.** Given an ICDFA$_\emptyset$ $A_\emptyset = \langle Q, \Sigma, \delta, q_0\rangle$, there is a bijection $\varphi : Q \mapsto [0, n[$ where $n = |Q|$.

Lemma 2.4.2 ([1])**.** The bijection φ along with bijection $\Pi : \Sigma \mapsto [0, k[$ where $k = |\Sigma|$ defines an isomorphism between the ICDFA$_\emptyset$ $\langle Q, \Sigma, \delta, q_0\rangle$ and $\langle [0, n[, [0, k[, \Delta, q_0\rangle$, where $\Delta(i, \sigma) = \varphi(\delta(\varphi^{-1}(i), \sigma))$.

Because the states of an ICDFA$_\emptyset$ have a unique order, the canonical string that represents the automaton is defined by:

$$\begin{aligned} &(s_i)_{i\in[0,kn[} && s_i \in [0, n[, \\ &s_i = \Delta(\lfloor i/k\rfloor, \Pi^{-1}(i \mod k)) && i \in [0, kn[. \end{aligned}$$

Lemma 2.4.3 ([1])**.** Let $A = \langle Q, \Sigma, \delta, q_0\rangle$ be an ICDFA$_\emptyset$, with $n = |Q|$, $k = |\Sigma|$ and let $(s_i)_{i\in[0,kn[}$ be its canonical string. Then every string satisfying rules R0, R1 and R2 represents an ICDFA$_\emptyset$ with n states over an alphabet of k symbols.

$$(\exists j \in [0, kn[)\ s_j = n - 1, \tag{R0}$$
$$(\forall m \in [2, n[)(\forall i \in [0, kn[)(s_i = m \Rightarrow (\exists j \in [0, i[)\ s_j = m - 1), \tag{R1}$$
$$(\forall m \in [1, n[)(\exists j \in [0, km[)s_j = m. \tag{R2}$$

It follows from the lemmas that there is a one-to-one mapping between strings satisfying R1 and R2 and non-isomorphic n-states k-symbols ICDFA$_\emptyset$s.

The flags of an automaton, concept introduced in [1], is the sequence: $(f_j)_{\ j\in[1,n]}$, that is, the index on the canonical string of the first occurrence of a state label j.

Theorem 2.4.1 (Th. 6 of [1])**.** Given k and n, the number of valid sequences $(f_j)_{\ j\in[1,n[}$, $F_{k,n}$, is given by:

$$F_{k,n} = \sum_{f_1=0}^{k-1} \sum_{f_2=f_1+1}^{2k-1} \cdots \sum_{f_{n-1}=f_{n-2}+1}^{k(n-1)-1} 1 = \binom{kn}{n} \frac{1}{(k-1)n+1} = C_n^{(k)}, \tag{2.1}$$

where $C_n^{(k)}$ are the (generalized) Fuss-Catalan numbers.

To generate all possible ICDFA$_\emptyset$ canonical strings, one just has to "fill the spaces" on the string for each configuration of flags $(f_j)_{j\in[1,n[}$, according to the following rules:

$$\begin{aligned}
&i < f_1 \Rightarrow s_i = 0,\\
&(\forall j \in [1, n-2])(f_j < i < f_{j+1} \Rightarrow s_i \in [0, j]),\\
&i > f_{n-1} \Rightarrow s_i \in [0, n[.
\end{aligned}$$

It follows that for each sequence of flags $(f_j)_{j\in[1,n]}$ the number of canonical strings is

$$\Pi_{j\in[1,n]} j^{f_j - f_{j-1}}.$$

Theorem 2.4.2 (Th. 8 of [1])**.** The number of strings $(s_i)_{i\in[0,kn[}$ representing ICDFA$_\emptyset$s with n states over an alphabet of k symbols is given by

$$B_{k,n} = \sum_{f_1=0}^{k-1} \sum_{f_2=f_1+1}^{2k-1} \cdots \sum_{f_{n-1}=f_{n-2}+1}^{k(n-1)-1} \prod_{j\in[1,n]} j^{f_j - f_{j-1}}, \tag{2.2}$$

where $f_n = kn$ and $f_0 = -1$.

Based on (2.1) and (2.2) it is possible to define a recursive form $N_{m,j}$ that counts the number of ICDFA$_\emptyset$s with prefix $s_0 s_1 \dots s_i$ with the first occurrence of symbol m in position j.

$$N_{n-1,j} = n^{nk-1-j} \qquad \text{with } j \in [n-2, (n-1)k[\tag{2.3}$$

$$N_{m,j} = \sum_{i=0}^{(m+1)k-j-2} (m+1)^i N_{m+1,j+i+1} \qquad \text{with } m \in [1, n-1], j \in [m-1, mk[\tag{2.4}$$

Theorem 2.4.3 (10 of [1])**.** Given the number of states n and the number of alphabet symbols k, the number of distinct ICDFA$_\emptyset$ $B_{k,n}$ can be expressed in function of $N_{m,j}$:

$$B_{k,n} = \sum_{l=0}^{k-1} N_{1,l}.$$

Random generation of n-state k-symbol ICDFA$_\emptyset$ up to isomorphism consists in:

- uniformly choosing a random integer from $[0, B_{k,n}[$.
- calculating the parameters m and j of all the recursive calls to (2.3), effectively extracting the values of the flags $(f_i)_{i\in[1,n]}$ of the canonical string of the ICDFA$_\emptyset$. This can be done in $O(n^2k)$ complexity using dynamic programming techniques when evaluating $N_{m,j}$.
- calculating the remaining non-flagged symbols of the canonical string using remainders of integer divisions. Refer to Algorithm 5 for the conversion of an integer to an ICDFA$_\emptyset$.

Algorithm 5 Converting an integer to n-state k-symbol ICDFA$_\emptyset$

procedure IntToFlags(m, $n = |Q|$, $k = |\Sigma|$)
  $s \leftarrow 1$
  for $i \in [1, n[$ **do**
    $j \leftarrow ik - 1$
    $p \leftarrow i^{j-f_{i-1}-1}$
    while $j \geq i - 1 \wedge m \geq ps \times N_{i,j}$ **do**
      $m \leftarrow m - ps \times N_{i,j}$
      $j \leftarrow j - 1$
      $p \leftarrow p/i$
    $s \leftarrow s \times i^{j-f_{i-1}-1}$
    $f_i \leftarrow j$
  return $(f_i)_{i\in[1,n]}$
procedure IntToIcdfa$_\emptyset$String(m, $n = |Q|$, $k = |\Sigma|$)
  $i \leftarrow kn - 1$
  $j \leftarrow n - 1$
  $f \leftarrow$ IntToFlags(m, n, k)
  while $m > 0 \wedge j > 0$ **do**
    while $m > 0 \wedge i > f_j$ **do**
      $s_i \leftarrow m \mod (j + 1)$
      $m \leftarrow m/(j + 1)$
      $i \leftarrow i - 1$
    $i \leftarrow i - 1$
    $j \leftarrow j - 1$
  return $(s_i)_{i\in[0,kn[}$

Although this not a generator for NFA, it deserves special attention because it serves as a base for the uniform random generator of an NFA class proposed in Chapter 3. This generator has the advantage of being efficient and uniformly generating up to isomorphism, and also counts the number of n-state k-symbol ICDFA$_\emptyset$ or ICDFA.

We provide some properties of the ICDFA from the FAdo implementation of the generator in Table 2.5. Our experiment consisted in sampling 10000 ICDFAs using the method described in this section, with: $minS$ - average number of states of the minimal DFA; $minT$ - average number of transitions of the minimal DFA; $usefulS$ - average number of reachable useful states; T - average number of transitions; $timeg$ - time in seconds required to sample the 10000 NFAs; $timep$ - time in seconds required to process the 10000 NFAs.

2.5 Comparative Analysis

From the Tables 2.1, 2.2, 2.4 and 2.5 one notices that the Markov chain quickly becomes intractable for high confidence level (10000 instances) experiments as either n grows or k increases.

We summarize the main common properties of the studied generators on Table 2.6. The

(n, k)	minS	minT	usefulS	T	timeg	timep
(5, 2)	4.46	8.92	4.85	9.58	0.22	1.24
(5, 3)	4.65	13.95	4.95	14.80	0.11	0.55
(5, 10)	4.77	47.66	5.00	50.00	0.36	1.65
(10, 2)	9.75	19.50	9.95	19.83	0.14	1.53
(10, 3)	9.95	29.86	9.99	29.97	0.27	1.91
(10, 10)	9.98	99.82	10.00	100.00	0.91	6.14
(15, 2)	14.80	29.59	14.97	29.90	0.24	3.31
(15, 3)	14.98	44.94	15.00	44.99	0.44	4.75
(15, 10)	15.00	150.00	15.00	150.00	2.07	16.19
(25, 2)	24.82	49.64	24.99	49.95	0.69	10.15
(25, 3)	24.99	74.97	25.00	75.00	1.20	14.87
(25, 10)	25.00	250.00	25.00	250.00	6.71	56.96

Table 2.5: DFA properties with uniform random generation up to isomorphism.

columns have the following interpretation: n - number of states of the automaton; m - number of symbols of the regular expression; k - size of the alphabet of the automaton; *Isomorphism* - whether the generator uniformly generates the automata up to isomorphism; *Fixed n* - whether the generator allows to fix the number of states of the resulting automaton; *Initially Connected* - whether all the generated automata are initially connected; *Time Complexity* - asymptotic time complexity of generating a single automaton; *NFA* - whether NFAs are generated or DFAs. We denote $\mathrm{MIX}(\mathbb{C}, n)$ the asymptotic mixing time of the Markov chain for a given class $\mathbb{C}$ of NFA and $\mathrm{AUT}(n)$ the asymptotic time complexity of n-state NFA automorphism counting problem.

	Isomorphism	Fixed n	Initially Connected	Time Complexity	NFA
Regular Expressions	No	No	Yes	$O(m^2)$	Yes
Bitstream	No	Yes	No. Except if density is enforced.	$O(n^2k)$	Yes
Markov chain	Yes	Yes	Depends on chosen class	$O(\mathrm{MIX}(\mathbb{C}, n) \cdot n^2k \cdot \mathrm{AUT}(n))$	Yes
ICDFA	Yes	Yes	Yes	$O(n^2k)$	No

Table 2.6: Comparison of properties of common NFA generators.

Chapter 3

Forward Injective Finite Automata

This class consists of initially connected NFAs, with a single initial state such that whenever unvisited states are reached from an already visited state in breadth-first search order, we must ensure that the set of transition labels from the current state to the newly reached states are pairwise distinct [11]. We denote NFAs with this property as *forward injective finite automata* (FIFA).

Given an NFA transition function $\delta(x, \sigma) : Q \times \Sigma \to 2^Q$, let $\delta(x) : Q \to 2^Q$ be the set of reached states from state x, i.e. , $\delta(x) = \cup_{\sigma \in \Sigma} \delta(x, \sigma)$. Let $\ell(x, y) : Q \times Q \to 2^\Sigma$ be the set of labels of transitions that reach state y from state x. Let $\ell_x(y) = \ell(x, y)$ for a fixed x. A FIFA is defined as:

Definition 1 ([11])**.** *Let $A = \langle Q, \Sigma, \delta, q_0, F \rangle$ be an initially-connected NFA and Π the bijection from 2^Σ to $[0, 2^k[$, induced by the order on Σ. Consider the breadth-first search traversal of A, induced by the bitmask order on 2^Σ, that starts in the initial state q_0. For each state $s \in Q$, let $S(s)$ be the set of states that are image of a transition starting from a state already visited by the BFS. The labels for the transitions departing from s to any state in $\delta(s) \setminus S(s)$ need to be unique. The automaton A is a* forward injective finite automaton *if it holds that:*

$$(\forall p, q \in \delta(s) \setminus S(s))(p \neq q \Rightarrow \ell(s, p) \neq \ell(s, q)). \tag{3.1}$$

This class of automata is expressive enough to recognize all regular languages, because deterministic automata trivially satisfy (3.1). However, not all (initially-connected) NFAs are FIFAs, e.g. on Figure 3.1.

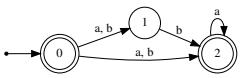


Figure 3.1: Non-FIFA initially-connected NFA.

Some experimental results suggest that, for alphabets of size at least 2, one can find a FIFA

equivalent to an NFA that is not much larger than the NFA. Because of the existence of uniform random generator for this class, one can use this model to obtain estimates of average performance of algorithms that manipulate NFAs.

3.1 A Canonical State Order for FIFAs

Given a FIFA it is possible to obtain a canonical state order φ through a breadth first traversal starting in the initial state and ordering the newly reached states according to the total order defined in 2^Σ. For that, one can disregard the set of final states and consider the semiautomaton $\text{FIFA}_\emptyset$. The canonical state order for a $\text{FIFA}_\emptyset$ can be computed through Algorithm 6, where Π is the bijection from 2^Σ to $[0, 2^k[$, *sorted* is a function that sorts integers in increasing order, and $\varphi : Q \to [0, n[$ is the computed bijection. Note that (at line 7) $\ell^{-1}_{\varphi^{-1}(s)}(\Pi^{-1}(j))$ is a single value by the injectivity of ℓ restricted to the newly seen states in a FIFA.

Algorithm 6 FIFA state order algorithm

```
1:  procedure StateOrder(FIFA_∅ ⟨Q, Σ, δ, q_0⟩)
2:      φ(q_0) ← 0
3:      i ← 0;  s ← 0
4:      do
5:          M ← sorted{ Π(S) | ∅ ≠ S = ℓ(φ^{-1}(s), q) ∧ q ∈ Q \ φ^{-1}([0, i]) }
6:          for j ∈ M do
7:              φ(ℓ^{-1}_{φ^{-1}(s)}(Π^{-1}(j))) ← i + 1
8:              i ← i + 1
9:          s ← s + 1
10:     while s < i
        return φ
```

Proposition 2 ([11])**.** *Let $A = \langle Q, \Sigma, \delta, q_0\rangle$ be a FIFA$_\emptyset$ with n states and $k = |\Sigma|$, there is a bijection $\varphi : Q \to [0, n[$ that defines an isomorphism between A and $\langle [0, n[, \Sigma, \delta', 0\rangle$ with $\delta'(i, \sigma) = \{\ \varphi(s) \mid s \in \delta(\varphi^{-1}(i), \sigma)\ \}$, for $i \in [0, n[$ and $\sigma \in \Sigma$.*

Proof. The proof follows the lines of a similar result for ICDFAs. Let φ be the function defined by the Algorithm 6. That φ is injective is trivial because different values are always assigned to different elements of Q. To prove that φ is surjective, let $q \in Q$. Because A is initially connected there exist a sequence $(q'_i)_{i\in[0,j]}$ of states and a sequence $(\sigma_i)_{i\in[0,j-1]}$ of symbols, for some $j < |Q|$, such that $q'_{m+1} \in \delta(q'_m, \sigma_m)$ for $m \in [0, j[$, $q'_0 = q_0$ and $q'_j = q$. We have $\varphi(q'_0) = 0$. For $m \in [0, j[$, let $\varphi(q'_m) = i_m$. Then either $q'_{m+1} \in \varphi^{-1}([0, i_m])$ or q'_{m+1} will be given the value $\varphi(q'_{m+1})$ in line 7 when $s = i_m$ and $j = \Pi(\ell(q'_m, q'_{m+1}))$. That there exists a unique value $q'_{m+1} = \ell^{-1}_{q'_m}(\Pi^{-1}(j))$ is ensured by the injectivity of ℓ restricted to the states newly seen in state q'_m (i.e satisfying (3.1)). Thus, one can conclude that $\varphi^{-1}([0, n[) = Q$. By the definition of δ', φ defines a (semi)automaton isomorphism. □

Throughout the chapter we will now consider $\text{FIFA}_\emptyset$ to have its states in their canonical order: $A = \langle [0, n[, \Sigma, \delta, 0 \rangle$.

3.2 Canonical String Representation

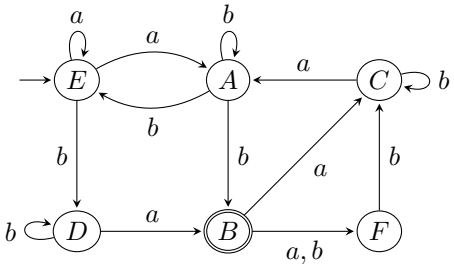
Let $A = \langle [0, n[, \Sigma, \delta, 0, F \rangle$ be a FIFA such that $\langle [0, n[, \Sigma, \delta, 0 \rangle$ is a $\text{FIFA}_\emptyset$. We can represent A by the canonical representation of its $\text{FIFA}_\emptyset$ concatenated with the bitmap of the state finalities. The canonical representation of a $\text{FIFA}_\emptyset$ is defined as follows.

Definition 2. *Given a FIFA$_\emptyset$ $\langle [0, n[, \Sigma, \delta, 0 \rangle$ with $|\Sigma| = k$, its canonical representation is a sequence $(r_i)_{i \in [0,n[}$ such that for each state i,*

$$r_i \quad = \quad s_{i,1} s_{i,2} \dots s_{i,m_i} u_{i,1} \dots u_{i,\overline{m}_i},$$

and where m_i is the number of previously seen states, $\overline{m}_i$ is the number of newly seen states, $s_{i,j} = \Pi(\ell(i, j-1))$ for $j \in [1, m_i]$, and $u_{i,j} = \Pi(\ell(i, m_i + j - 1))$ for $j \in [1, \overline{m}_i]$. This means that, for each state i, $s_{i,j}$ correspond to the sets of transitions to states already seen (back transitions) *and $u_{i,j}$ correspond to the sets of transitions to newly seen states from state i* (forward transitions).

Example 3.2.1. Consider the following FIFA on the left.



φ	A	B	C	D	E	F
	1	3	4	2	0	5

.

Let $\Pi(\emptyset) = 0$, $\Pi(\{a\}) = 1$, $\Pi(\{b\}) = 2$ and $\Pi(\{a, b\}) = 3$. The state renaming according to the induced order on the states is given above. The canonical string $(r_i)_{i \in [0,5]}$ for the corresponding $\text{FIFA}_\emptyset$ is

$$\underbrace{[1][1,2]}_{E}\ \underbrace{[2,2,0][2]}_{A}\ \underbrace{[0,0,2,1][]}_{D}\ \underbrace{[0,0,0,0][1,3]}_{B}\ \underbrace{[0,1,0,0,2,0][]}_{C}\ \underbrace{[0,0,0,0,2,0][]}_{F},$$

where the transitions for each state are as indicated. The FIFA can be represented by its semiautomaton canonical string with the state finalities appended. Thus, this FIFA canonical string is

$$[1][1,2][2,2,0][2][0,0,2,1][][0,0,0,0][1,3][0,1,0,0,2,0][][0,0,0,0,2,0][][0,0,0,1,0,0].$$

Lemma 3.2.1. Let $A = \langle [0, n[, \Sigma, \delta, 0 \rangle$ be a $\text{FIFA}_\emptyset$ with $k = |\Sigma|$. Let $(r_i)_{i \in [0,n[}$ with $r_i = s_{i,1} s_{i,2} \cdots s_{i,m_i} u_{i,1} u_{i,2} \cdots u_{i,\overline{m}_i}$ be the canonical representation for A as given above. Then the

following rules are satisfied:

$$s_{i,j} \in [0, 2^k[, \qquad \forall i \in [0, n[, \forall j \in [1, m_i], \tag{F1}$$

$$u_{i,j} \in [1, 2^k[, \qquad \forall i \in [0, n[, \forall j \in [1, \overline{m}_i], \tag{F2}$$

$$j < l \Rightarrow u_{i,j} < u_{i,l}, \qquad \forall i \in [0, n[, \forall j, l \in [1, \overline{m}_i], \tag{F3}$$

$$m_0 = 1, \tag{F4}$$

$$m_i = m_{i-1} + \overline{m}_{i-1}, \qquad \forall i \in [1, n[, \tag{F5}$$

$$i < m_i \leq n, \qquad \forall i \in [1, n[, \tag{F6}$$

$$\overline{m}_{n-1} = 0. \tag{F7}$$

Proof. The rule (F1) describes how transitions to previously seen states are represented, i.e. $s_{i,j} = \Pi(\ell(i, j-1))$, possibly with $s_{i,j} = 0$. The rule (F2) considers the transitions from state i to states $t_{i,j} = m_i + j - 1$ visited for the first time in i, which implies that $\ell(i, t_{i,j}) \neq \emptyset$. Consequently, $u_{i,j} \in [1, 2^k[$. For representation purposes, rule (F3) states that the set of states visited for the first time is represented with its transitions sorted in ascending order. This is a representation choice that ensures that all $u_{i,j}$ are distinct. Rule (F4) is obvious as one starts at state 0 and thus 0 is the only seen state. Rule (F5) is a direct consequence of the definition of m_i in Equation (2), and implies that $m_i = 1 + \sum_{j=0}^{i-1} \overline{m}_j$ for $i \in [1, n[$. Rules (F6) and (F7) ensures that all states are seen, and that the FIFA$_\emptyset$ is initially connected. It is a consequence of the definition of φ and also ensures that a state must be seen before its representation is given. Rule (F6) implies that $m_{n-1} = n$. By contradiction, suppose that there is i such that $m_i \leq i$, for $i \in [1, n[$. Then, there exists $1 \leq j \leq i$ such that j is not accessible from 0 in paths that use states only in $[0, i[$. But that contradicts the definition of φ. □

Lemma 3.2.2. Every string $(r_i)_{i \in [0,n[}$ satisfying rules (F1)-(F7) represents a FIFA$_\emptyset$ with states $[0, n[$ over an alphabet of k symbols.

Proof. There is at least one transition reaching each state in $[1, n[$ and there is at least one transition from the state 0 to state 1. The transition function is defined by $\ell(i, j-1) = \Pi^{-1}(s_{i,j})$ for $i \in [0, n[$ and $j \in [1, m_i]$, and $\ell(i, m_i + j - 1) = \Pi^{-1}(u_{i,j})$ for $i \in [0, n[$ and $j \in [1, \overline{m}_i]$. The proof that the FIFA$_\emptyset$ is initially connected is analogous to the one in Lemma 3.2.1. □

From these lemmas the following theorem holds.

Theorem 3.2.1. For each $n > 0$ and $k > 0$, there is a one-to-one mapping from sequences $(r_i)_{i \in [0,n[}$ satisfying rules (F1)–(F7) and non-isomorphic FIFA$_\emptyset$s with n states over an alphabet of k symbols.

The canonical form for FIFA$_\emptyset$ is not a simple extension of the one for ICDFA$_\emptyset$s reviewed in Section 2.4 for several reasons. One needs to consider instead of the alphabet its power set, there are no restrictions for transitions to already seen states, and transitions to newly seen states must have different labels. However, the first occurrences of each state satisfy exactly the

same rules (over an alphabet of 2^k symbols) observed in the canonical representation of ICDFA$_\emptyset$s. This will be made evident in the next section.

3.3 Counting FIFAs

Our aim is to enumerate (exactly generate) and count all the nonisomorphic FIFA$_\emptyset$ with n states and k symbols. This will also allow us to obtain a uniform random generator for the class of FIFAs.

Let $\Psi : [0, n[\times[1, 2^k[\to [0, n(2^k - 1)[$, be defined by $\Psi(i, j) = i(2^k - 1) + j - 1$. The mapping Ψ is a bijection with $\Psi^{-1}(p) = \left(\left\lfloor p/(2^k - 1)\right\rfloor, (p \bmod (2^k - 1)) + 1\right)$.

Let $(r_i)_{i\in[0,n[}$ be a sequence satisfying rules (F1)–(F7), thus, representing a FIFA$_\emptyset$. Let us denote by *flag of a state* $t \in [1, n[$ the pair $(i, u_{i,j})$, occurring in state i, such that $t = m_i + j - 1$ (and $\ell(i, t) = \Pi^{-1}(u_{i,j})$).

According to (F3), if in a state i two different flags $(i, u_{i,j})$ and $(i, u_{i,l})$ occur, we know that $j < l \Rightarrow u_{i,j} < u_{i,l}$. For the sake of readability, given $t \in [1, n[$, we denote by (i_t, u_t) its flag, and let $\Psi(i_t, u_t) = f_t$. Then, by (F3), one has

$$(\forall t \in [2, n[)(i_t = i_{t-1} \Rightarrow u_t > u_{t-1}) \vee (i_t > i_{t-1}),$$

which implies

$$(\forall t \in [2, n[)(f_t > f_{t-1}), \tag{G1}$$
$$(\forall t \in [1, n[)(f_t < t(2^k - 1)). \tag{G2}$$

Rules (G1)–(G2) are the ones satisfied by the positions of the first occurrence of a state in the canonical strings for ICDFA$_\emptyset$s, considering k instead of $2^k - 1$ in rule (G2). The following theorem computes the number of sequences of flags that are allowed.

Proposition 3 (Theorem 6 of [1]). *Given $k > 0$ and $n > 0$, the number of sequences $(f_t)_{t\in[1,n[}$, $F_{2^k-1,n}$, is given by:*

$$F_{2^k-1,n} = \sum_{f_1=0}^{2^k-1-1} \sum_{f_2=f_1+1}^{2(2^k-1)-1} \cdots \sum_{f_{n-1}=f_{n-2}+1}^{(2^k-1)(n-1)-1} 1 = C_n^{(2^k-1)},$$

where $C_n^{(2^k-1)} = \binom{n(2^k-1)}{n} \frac{1}{(2^k-2)n+1}$ are the generalized Fuss-Catalan numbers.

Example 3.3.1. For the FIFA of Example 3.2.1, $((0, 1), (0, 2), (1, 2), (3, 1), (3, 3))$ is the sequence of flags and $(f_t)_{t\in[1,5]} = (0, 1, 4, 9, 11)$.

Given a sequence of flags $(f_t)_{t\in[1,n[}$, the set of possible canonical strings that represent FIFA$_\emptyset$s can be easily enumerated: each state i has unconstrained transitions for states already seen (m_i)

and has the transitions to new states given by the flags occurring in its description (forward transitions). It is trivial that one can obtain $Q_{i+1} \setminus Q_i$ from the flags and hence m_i. The number of canonical strings with a given sequence of flags is given by

$$\prod_{i \in [0,n[} (2^k)^{m_i}. \tag{3.2}$$

Thus, the following theorem holds.

Theorem 3.3.1. The total number of FIFA$_\emptyset$s with n states over an alphabet of k symbols is

$$b_{k,n} \quad = \quad \sum_{f_1=0}^{2^k-1-1} \sum_{f_2=f_1+1}^{2(2^k-1)-1} \cdots \sum_{f_{n-1}=f_{n-2}+1}^{(n-1)(2^k-1)-1} \prod_{i \in [0,n[} (2^k)^{m_i},$$

where $m_i = 1 + \sum_{j=0}^{i-1} \overline{m}_j$ and $\overline{m}_j = |\{\ f_t \mid i_t = j\}|$ for $i \in [0, n[$ and $j \in [1, n[$.

This can be adapted for the exact generation/enumeration of all canonical representations. Each FIFA$_\emptyset$ corresponds to a number between 1 and $b_{k,n}$. In Table 3.1 we present the values of $b_{k,n}$ for $n \in [2, 7]$ and $k \in [2, 3]$. An equivalent recursive definition for $b_{k,n}$ is given in the next section for uniform random generation.

Table 3.1: Values of $b_{k,n}$

n	$k = 2$	$k = 3$
2	192	3584
3	86016	56885248
4	321912832	32236950781952
5	10382009696256	738091318939425439744
6	3073719939819896832	733871593861464877408622477312
7	871581830440515993285427 2	326867227491799792314941447869937011916 80

Corollary 3.3.1. The number of non-isomorphic FIFAs with n states and k alphabetic symbols is $B_{k,n} = b_{k,n} 2^n$.

$$B_{k,n} \quad = \quad b_{k,n} 2^n. \tag{3.3}$$

3.4 Uniform Random Generation

The canonical representation for FIFAs allows an easy uniform random generation for this class of automata. Given the number of flags occurring in a prefix of a canonical string we count the number of valid suffixes. To count the number of automata with a given prefix a recursive counting formula for FIFA$_\emptyset$ is needed. With these partial values, we can reconstruct any FIFA$_\emptyset$ by knowing its number, which varies from 1 to $b_{k,n}$. The process of uniform randomly generating

a FIFA consists, thus, in four steps: creation of a table with partial counts for each prefix; uniformly sample a number between 1 and $b_{k,n}$; construct the $\text{FIFA}_\emptyset$ representation using the table; random generation of values from 0 to $2^n - 1$ for the state finalities and return the FIFA.

1. creation of a table with partial counts for each prefix;
2. uniformly sample a number from 1 to $b_{k,n}$;
3. reconstruct the $\text{FIFA}_\emptyset$ using the table;
4. random generation of values from 0 to $2^n - 1$ for the state finalities and return the FIFA.

Let m be the number of already seen states for the state i of a canonical string of a $\text{FIFA}_\emptyset$. We count the number $N_{m,i}$ of $\text{FIFA}_\emptyset$s for each value of m and i. This gives us the following recursive formula for fixed n and k:

$$\begin{array}{lcll} N_{m,i} & = & (2^k)^m \sum_{j=0}^{n-m} \binom{2^k-1}{j} N_{m+j,i+1}, & m \in [1,n], i \in [0,m[, \\ N_{m,i} & = & 0, & m \in [1,n], i \notin [0,m[, \\ N_{n,n} & = & 1. & \end{array}$$

Proposition 4. *$b_{k,n} = N_{1,0}$, for all $k \geq 1$ and $n \geq 1$.*

Proof. Immediate consequence of the canonical representation and Theorem 3.3.1. □

Proposition 5. *Algorithm 7 presents a uniform random generator for a $\text{FIFA}_\emptyset$ with n states and k symbols.*

To obtain a random FIFA from the $\text{FIFA}_\emptyset$ we can generate a random number from $[0, 2^n[$ and reconstruct the state finalities according to the corresponding choice. Using dynamic programming techniques it is possible to generate a table indexed by the values of m and i ($N_{m,i}$) with time complexity $O(n^3 \log((2^k)^{n^2})) = O(n^5 k)$. The amount of memory used is $O(n^4 k)$, and this is a limiting factor for the dimension of the $\text{FIFA}_\emptyset$ being generated. This is justified by the huge number of $\text{FIFA}_\emptyset$s for a given n and k. For example, $b_{2,100}$ is greater than 10^{11531}. In Table 3.2 we present the execution times for the generation of 10000 FIFAs for $n \in \{1, 20, 30, 50, 75, 100\}$ and $k \in \{1, 2, 3, 4\}$, using Python 2.7 interpreted by Pypy, with a Intel Xeon CPU X5550 at 2.67GHz. Comparing with some experiments presented by Héam and Joly [12, Table 1], these times correspond, approximately, to the generation of a single NFA.

3.5 Converting an NFA into a FIFA

In this section we discuss a process of converting an arbitrary NFA to a FIFA and the asymptotic time complexity bounds of such a procedure. The algorithm has an NFA as input and outputs an equivalent FIFA. It is based on the subset construction for NFA determinisation, with addition

Algorithm 7 Random $\mathrm{FIFA}_\emptyset$ algorithm.

procedure RANDOMFIFA(n, k)
  $r \leftarrow Random(0, N_{1,0} - 1)$ ▷ number of the $\mathrm{FIFA}_\emptyset$
  $m_0 \leftarrow 1$
  for $q \in [0, n-1[$ **do** ▷ reconstruct $\mathrm{FIFA}_\emptyset$ flags
    $\overline{m}_q \leftarrow 0$
    $ac \leftarrow 0$
    while $ac \leq r$ **do**
      $\overline{m}_q \leftarrow \overline{m}_q + 1$
      $ac \leftarrow (2^k)^{m_q} \binom{2^k-1}{\overline{m}_q} N_{q,m_q+\overline{m}_q}$
  $m_{q+1} \leftarrow m_q + \overline{m}_q$
  $b \leftarrow r \bmod (2^k)^{m_q}$ ▷ number representing back transitions
  $r \leftarrow (r-b)/(2^k)^{m_q}$
  $f \leftarrow r - (2^k)^{m_q} \sum_{i=0}^{m_q-1} \binom{2^k-1}{i} N_{m_q+i,q+1}$
  $f \leftarrow f/(2^k)^{m_q}$ ▷ number representing forward transitions
  $r \leftarrow (f - f \bmod \binom{2^k-1}{\overline{m}_q})/\binom{2^k-1}{\overline{m}_q}$
  for $p \in [1, m_q]$ **do** ▷ reconstruct back transitions
    $s_{q,p} \leftarrow b \mod 2^k$
    $b \leftarrow b/2^k$
  if $\overline{m}_q \neq 0$ **then**
    $c \leftarrow \binom{2^k-1}{\overline{m}_q}$
    $t \leftarrow f \bmod c$
    $f \leftarrow f/c$
    for $p \in [1, \overline{m}_q]$ **do**
      $u_{q,p} \leftarrow t \bmod (2^k - 1)$
      $t \leftarrow \lfloor t/(2^k-1) \rfloor$
for $p \in [1, n]$ **do** ▷ reconstruct back transitions
  $s_{n-1,p} \leftarrow r \mod 2^k$
  $r \leftarrow r/2^k$
return $(s_{i,1}s_{i,2}\cdots s_{i,m_i}u_{i,1}u_{i,2}\cdots u_{i,m_i})_{i\in[0,n[}$

of an heuristic that attempts to create back transitions whenever possible. This gives us a FIFA that is not only forward injective but also forward deterministic. It may be also possible to add an heuristic for nondeterministic forward injective transitions or to have other procedures that are not based on the subset construction. However this one had a good performance in our experiments.

We present a method of converting an NFA $\langle Q, \Sigma, \delta, I, F\rangle$ into a FIFA $\langle Q', \Sigma, \delta', I, F'\rangle$ such that: $Q' \subseteq 2^Q$, $\delta' \subseteq Q' \times \Sigma \times 2^{Q'}$, $F' = \{\ S \mid S \in Q', S \cap F \neq \emptyset\ \}$ and $\delta'(S, a)$ satisfies (3.1) by having at most one forward transitions for all $S \in Q'$ and $a \in \Sigma$. This method in implemented in Algorithm 8.

The following lemmas apply to Algorithm 8:

Lemma 3.5.1. For all $a \in \Sigma$ and $S \in Q'$, δ' contains at most one forward transition (to state $P \in Q'$) and a series of transitions for m previously seen states $R_1, \ldots, R_m$ with $R_i \in Q'$.

Table 3.2: Execution times for the generation of 10000 random FIFA.

Times	$k=1$	$k=2$	$k=3$	$k=4$
$n=10$	0.77s	1.05s	0.95s	8.59s
$n=20$	1.06s	2.33s	3.13s	3.96s
$n=30$	1.15s	5.01s	7.38s	9.52s
$n=50$	2.84s	16.86s	26.64s	40.43s
$n=75$	7.11s	47.62s	71.92s	91.70s
$n=100$	15.86s	100.25s	156.24s	202.41s

Algorithm 8 An NFA to FIFA algorithm

```
1:  procedure NfaToFifa(NFA ⟨Q, Σ, δ, I, F⟩)
2:      Q' ← {I}
3:      w ← {I}                                          ▷ states to be processed
4:      while w ≠ ∅ do
5:          S ← Pop(w)                                   ▷ popping S from w
6:          for σ ∈ Σ do
7:              P ← ∅
8:              δ'(S, σ) ← ∅
9:              for s ∈ S do
10:                 P ← P ∪ δ(s, σ)
11:             for R ∈ sortedBySizeDesc(Q') do
12:                 if R ⊆ P then                        ▷ nondeterministic back transitions
13:                     δ'(S, σ) ← δ'(S, σ) ∪ {R}
14:                     P ← P \ R
15:             if P ≠ ∅ then                            ▷ forward transition
16:                 Q' ← Q' ∪ {P}
17:                 w ← w ∪ {P}
18:                 δ'(S, σ) ← δ'(S, σ) ∪ {P}
19:     return FIFA ⟨Q', Σ, δ', I, {S ∩ F ≠ ∅, S ∈ Q'}⟩
```

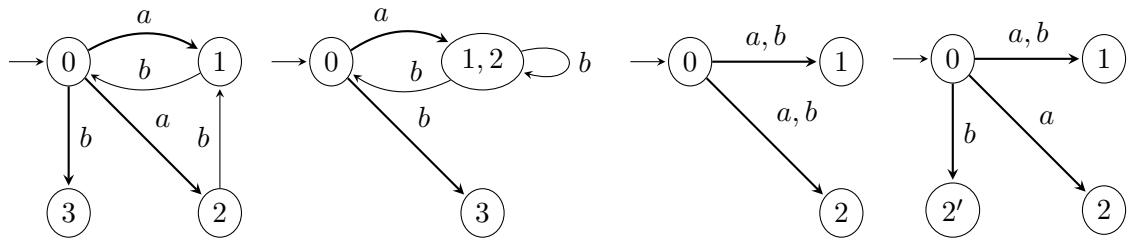
Proof. Direct consequence of δ' constructed in lines 11-14 in Algorithm 8 satisfying (3.1). □

Lemma 3.5.2. The following equality holds for all $R \in Q'$ and $a \in \Sigma$:

$$\delta(R,a) = \bigcup_{S \in \delta'(R,a)} S \tag{3.4}$$

Proof. By definition, $\delta : 2^Q \times \Sigma \mapsto 2^Q$ is:

$$\begin{aligned} \delta(R,a) &= \bigcup_{s \in R} \delta(s,a), && R \subseteq Q, a \in \Sigma \\ &= R_1 \cup \ldots \cup R_m \cup P, && \text{according to Lemma 3.5.1.} \end{aligned}$$



Furthermore, for $\delta' : Q' \times \Sigma \mapsto 2^{Q'}$ we have:

$$\delta'(R, a) = \{R_1, \ldots, R_m, P\}, \qquad R \in Q', a \in \Sigma.$$

□

We are now capable of proving the following lemma:

Lemma 3.5.3. The following equality holds for all $R \in Q'$ and $w \in \Sigma^\star$:

$$\delta(R, w) = \bigcup_{S \in \delta'(\{R\}, w)} S, \qquad R \in Q', w \in \Sigma^\star. \tag{3.5}$$

Proof. The proof will follow by induction on the size of the word w.

Base case (word = ε) By definition of the extension of δ to words and sets: $\delta(R, \varepsilon) = \{R\}$. We also have that $\bigcup_{S \in \delta'(\{R\}, \varepsilon)} S = \bigcup_{S \in \{R\}} S = \{R\}$, because δ' naturally extends to $2^{Q'} \times \Sigma^\star \mapsto 2^{Q'}$.

Induction step (word = wa)

$$\begin{aligned}
\delta(R, wa) &= \delta(\delta(R, w), a) \\
&= \delta\left(\bigcup_{S \in \delta'(\{R\}, w)} S, a\right) \\
&= \bigcup_{S \in \delta'(\{R\}, w)} \delta(S, a) \\
&= R_1 \cup \ldots \cup R_m \cup P, && R_i, P \in \delta'(\delta'(\{R\}, w), a) \\
&= R_1 \cup \ldots \cup R_m \cup P, && R_i, P \in \delta'(\{R\}, wa) \\
&= \bigcup_{S \in \delta'(\{R\}, wa)} S.
\end{aligned}$$

□

Theorem 3.5.1. Let A be an NFA $A = \langle Q, \Sigma, \delta, I, F \rangle$ and A' be the constructed FIFA $A' = \langle Q', \Sigma, \delta', I, F' \rangle$:

$$L(A) = L(A').$$

Proof.

$$
\begin{aligned}
w \in L(A) &\Rightarrow \delta(\{q_0\}, w) \cap F \neq \emptyset \\
&\Rightarrow \bigcup_{S \in \delta'(\{q_0'\}, w)} S \cap F \neq \emptyset && \text{by Lemma 3.5.3} \\
&\Rightarrow (\exists S \in Q')(S \in \delta'(\{q_0'\}, w) \wedge S \cap F \neq \emptyset) \\
&\Rightarrow (\exists S \in \delta'(\{q_0'\}, w))S \in F' && \text{by definition of } F' \\
&\Rightarrow w \in L(A'), \\
w \in L(A') &\Rightarrow \delta'(\{q_0'\}, w) \cap F' \neq \emptyset \\
&\Rightarrow (\exists S \in \delta'(\{q_0'\}, w))S \in F' \\
&\Rightarrow (\exists S \in \delta'(\{q_0'\}, w))S \cap F \neq \emptyset && \text{by definition of } F' \\
&\Rightarrow \left(\exists R \subseteq \bigcup_{S \in \delta'(\{q_0'\}, w)} S\right) R \cap F \neq \emptyset && \text{by Lemma 3.5.3} \\
&\Rightarrow \left(\bigcup_{S \in \delta'(\{q_0'\}, w)} S\right) \cap F \neq \emptyset \\
&\Rightarrow \delta(\{q_0\}, w) \cap F \neq \emptyset \\
&\Rightarrow w \in L(A).
\end{aligned}
$$

□

Proposition 6. *The procedure* NfaToFifa *in Algorithm 8 computes a FIFA equivalent to a given NFA.*

Proof. Algorithm 8 is the same as the subset construction for NFA determinisation, with addition of an heuristic that attempts to create back transitions whenever possible, in lines 11-14, by setting for a given reached subset S of Q and a symbol a of Σ the constructed $\delta'(S, a) = \{R_i, \ldots, R_m, P\}$ where:

$$
\begin{aligned}
R_i \cap R_j &= \emptyset && i, j \in \{1, \ldots, m\}, \\
R_i \cap P &= \emptyset && i \in \{1, \ldots, m\}, \\
\delta(S, a) &= \bigcup_{R \in \delta'(S,a)} R && S \in Q',\ a \in \Sigma.
\end{aligned}
$$

It follows that from the construction in Lemma 3.5.1 and Theorem 3.5.1 that the constructed automaton will be a FIFA with at most one newly reached state for each $s \in \Sigma$ and source state $S \in Q'$ and will recognize the same language as the input NFA.

□

This algorithm has time complexity $O(|\Sigma||Q|2^{2|Q|})$, which is justified by $|Q'|$ having space complexity $O(2^{|Q|})$, due to the determinisation based algorithm. The $|\Sigma|$ factor comes from the

outer loop in line 6 and the $|Q|$ factor comes from the comparison based algorithm for sorting Q' with time complexity $O(2^{|Q|}\log(2^{|Q|})) = O(|Q|2^{|Q|})$. The extra $2^{|Q|}$ factor comes from the maximum size of the working queue w. It is an open problem whether these bounds are tight for this algorithm.

3.6 Experimental Results

The algorithm defined in the previous section was implemented within the FAdo package [10]. We performed some experiments to compare the sizes of the input and output automata. The input NFAs were obtained from partial derivative construction over uniform random generated regular expressions, for a fixed (standard) grammar, of a given syntactic tree size m over an alphabet of k symbols. The conversion method used was the partial derivative automata [2]. For each m and k, 10000 random regular expressions were generated to ensure a 95% confidence level within a 1% error margin [16, pp. 38–41]. For each sample, we calculated the minimal, the average and the maximum sizes of the obtained automata. For each partial derivative automaton (PD) we applied the algorithm NFAtoFIFA and obtained a FIFA (FIFA). We also computed the DFA obtained by determinisation of PD, by the usual subset construction, (DT), and the minimal DFA (MIN). Results for $m \in \{50, 100, 250, 500\}$ and $k \in \{2, 3, 10\}$ are presented in Table 3.3. In general the FIFA computed is not much larger than the PD, although the determinised automata can be significantly larger.

Table 3.3: State complexities of automata where m, syntactic size of RE; k, size of alphabet; PD, size of partial derivative NFA; DT size of DFA from PD; MIN, size of minimal DFA; FIFA, size of FIFA from PD.

m, k	Type	min	avg	max
50, 2	FIFA	3	10.1684	25
	DT	3	10.1338	62
	MIN	1	5.0762	51
	PD	3	10.6904	19
100, 2	FIFA	3	19.2113	46
	DT	3	19.7193	158
	MIN	1	6.2814	116
	PD	9	20.0239	30
250, 2	FIFA	9	48.3731	107
	DT	12	185.6424	1120
	MIN	1	7.0256	630
	PD	34	47.998	66
500, 2	FIFA	35	99.8889	189
	DT	13	186.1518	6451
	MIN	1	6.8369	745
	PD	72	94.6422	124

m, k	Type	min	avg	max
50, 3	FIFA	3	12.4948	25
	DT	3	13.9339	56
	MIN	1	9.1225	41
	PD	4	11.6522	19
100, 3	FIFA	5	24.3173	45
	DT	5	34.289	166
	MIN	1	18.2094	148
	PD	11	21.8518	32
250, 3	FIFA	23	60.7792	110
	DT	12	185.6424	1586
	MIN	1	59.0185	988
	PD	34	52.5888	70
500, 3	FIFA	42	122.2247	198
	DT	37	1143.2134	11687
	MIN	1	92.8985	2343
	PD	79	103.6871	126

m, k	Type	min	avg	max
50, 10	FIFA	6	14.1252	24
	DT	7	15.3764	33
	MIN	1	13.8138	30
	PD	6	13.3556	21
100, 10	FIFA	14	27.4189	43
	DT	15	32.5608	66
	MIN	1	28.7841	58
	PD	13	25.2294	36
250, 10	FIFA	44	67.8454	99
	DT	55	102.6683	329
	MIN	1	88.8932	253
	PD	41	60.8428	78
500, 10	FIFA	102	135.1439	170
	DT	128	277.5053	1122
	MIN	1	237.4012	869
	PD	94	120.0465	150

Table 3.4: Ratio of PD NFA that are FIFA.

	$m = 25$	$m = 50$	$m = 75$	$m = 100$
$k = 2$	0.359	0.257	0.242	0.232
$k = 4$	0.570	0.429	0.406	0.369

Chapter 4

Conclusion

We presented some of the most common random generators of deterministic and non-deterministic finite automata. It can be seen on Table 2.6 that there is a trade off in asymptotic time complexity and universe of the automata class being generated.

Our research resulted in the classification of a specific NFA class, named FIFA, that would provide us all of common properties taken into consideration on Table 2.6. The n-state k-symbol automata in this class can be counted, randomly generated up to isomorphism and their isomorphism can be tested in polynomial time, assuming a fixed k.

Experimental results on NFA operations located on Table 3.3 show that the average increase on number of states in determinizing an NFA and determinizing the FIFA constructed from the same NFA is very similar. These results support our hypothesis of FIFA being a good candidate for significant experimental average case analysis of operations in NFA, with random generation of the sample being done in useful time.

4.1 Future Work

Future work includes estimating asymptotic bounds for the number of n-state k-symbol FIFAs, as Table 3.1 suggests this number quickly becomes intractable.

Upper bounds for the size of the minimal FIFAs for a given language will be of major interest. State complexity results on the operations with FIFAs is also an open and important problem, as it would be a key step in providing a formal argument on the validity of using FIFAs to empirically estimate average case properties of NFAs.

References

[1] Marco Almeida, Nelma Moreira, and Rogério Reis. Enumeration and generation with a string automata representation. *Theoretical Computer Science*, 387(2):93–102, 2007. ISSN: 03043975. doi:10.1016/j.tcs.2007.07.029.

[2] Valentin Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *Theoretical Computer Science*, 155(2):291 – 319, 1996. ISSN: 0304-3975. doi:10.1016/0304-3975(95)00182-4.

[3] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2009. ISBN: 0521424267, 9780521424264.

[4] László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In *Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing*, STOC '16, pages 684–697, New York, NY, USA, 2016. ACM. ISBN: 978-1-4503-4132-5. doi:10.1145/2897518.2897542.

[5] Frédérique Bassino and Cyril Nicaud. Enumeration and random generation of accessible automata. *Theoretical Computer Science*, 381(1-3):86–104, 2007. ISSN: 03043975. doi:10.1016/j.tcs.2007.04.001.

[6] Kellogg S. Booth. Isomorphism testing for graphs, semigroups, and finite automata are polynomially equivalent problems. *SIAM J. Comput.*, 7(3):273–279, 1978. doi:10.1137/0207023.

[7] Sabine Broda, António Machiavelo, Nelma Moreira, and Rogério Reis. Average Size of Automata Constructions from Regular Expressions. *Bulletin of the EATCS*, 116, 2015.

[8] Jean-Marc Champarnaud, Georges Hansel, Thomas Paranthoën, and Djelloul Ziadi. Random Generation Models for NFA's. *J. Autom. Lang. Comb.*, 9(2-3):203–216, September 2004. ISSN: 1430-189X.

[9] W.G. Cochran. *Sampling Techniques, 3Rd Edition*. A Wiley publication in applied statistics. Wiley India Pvt. Limited, 2007. ISBN: 9788126515240.

[10] Project FAdo. FAdo: tools for formal languages manipulation. http://fado.dcc.up.pt, Access date:1.3.2018.

[11] Miguel Ferreira, Nelma Moreira, and Rogério Reis. Forward injective finite automata: Exact and random generation of nonisomorphic nfas. In *Descriptional Complexity of Formal Systems - 20th IFIP WG 1.02 International Conference, DCFS 2018, Halifax, NS, Canada, July 25-27, 2018, Proceedings*, pages 88–100, 2018. doi:10.1007/978-3-319-94631-3_8.

[12] Pierre-Cyrille Héam and Jean-Luc Joly. On the uniform random generation of non deterministic automata up to isomorphism. In Frank Drewes, editor, *Implementation and Application of Automata*, pages 140–152, Cham, 2015. Springer International Publishing. ISBN: 978-3-319-22360-5. doi:10.1007/978-3-319-22360-5_12.

[13] Harald Andrés Helfgott. Isomorphismes de graphes en temps quasi-polynomial (d'après Babai et Luks, Weisfeiler-Leman...). *Séminaire Bourbaki*, 2017.

[14] John E. Hopcroft. An n log n algorithm for minimizing states in a finite automaton. In *The Theory of Machines and Computations*, pages 189–196, 1971. ISBN: 9780124177505. doi:10.1016/B978-0-12-417750-5.50022-1.

[15] John E. Hopcroft and Jeffrey D. Ullman. *Introduction To Automata Theory, Languages, And Computation*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1990. ISBN: 020102988X.

[16] Donald E. Knuth. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997. ISBN: 0-201-89684-2.

[17] Dexter Kozen. On kleene algebras and closed semirings. In Branislav Rovan, editor, *Mathematical Foundations of Computer Science 1990*, pages 26–47, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg. ISBN: 978-3-540-47185-1. doi:10.1007/BFb0029594.

[18] Dexter Kozen. A completeness theorem for kleene algebras and the algebra of regular events. *Information and Computation*, 110(2):366–390, 1994. ISSN: 10902651. doi:10.1006/inco.1994.1037.

[19] Ted Leslie. Efficient Approaches to Subset Construction. Technical report, University of Western Ontario, 1995.

[20] E. M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. In *21st Annual Symposium on Foundations of Computer Science (sfcs 1980)*, pages 42–49, Oct 1980. doi:10.1109/SFCS.1980.24.

[21] Harry G. Mairson. Generating words in a context-free language uniformly at random. *Information Processing Letters*, 49(2):95–99, 1994. ISSN: 00200190. doi:10.1016/0020-0190(94)90033-7.

[22] A. R. Meyer and M. J. Fischer. Economy of Description by Automata, Grammars, and Formal Systems. *Proceedings of the 12th Annual Symposium on Switching and Automata Theory (Swat 1971)*, pages 188–191, 1971. doi:10.1109/SWAT.1971.11.

[23] Gary L. Miller. Graph isomorphism, general remarks. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, STOC '77, pages 143–150, New York, NY, USA, 1977. ACM. doi:10.1145/800105.803404.

[24] Uwe Schöning. Graph isomorphism is in the low hierarchy. In Franz J. Brandenburg, Guy Vidal-Naquet, and Martin Wirsing, editors, *STACS 87*, pages 114–124, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg. ISBN: 978-3-540-47419-7. doi:10.1007/BFb0039599.

[25] Lynette van Zijl, John-Paul Harper, and Frank Olivier. The MERLin Environment Applied to $\star$-NFAs. In Shen Yu and Andrei Păun, editors, *Implementation and Application of Automata*, pages 318–326, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. ISBN: 978-3-540-44674-3. doi:10.1007/3-540-44674-5_28.